UNIVERSIDAD NACIONAL DE PIURA

Facultad de Ingeniería Industrial

Escuela Profesional de Ingeniería Informática

TESIS

"PROPONER UNA ARQUITECTURA DE SOFTWARE PARA EL DESARROLLO DE APLICACIONES EMPRESARIALES USANDO EL LENGUAJE DE PROGRAMACIÓN ESTRUCTURADO RPG EMPLEANDO ESTÁNDARES DE DISEÑO DE SOFTWARE."

Presentada Por:

Br. Nelson Abel Barranzuela Iman

PARA OPTAR EL TITULO PROFESIONAL DE INGENIERO INFORMATICO

Línea de investigación:

Informática, Electrónica y Telecomunicaciones

Sublínea de investigación:

Computación

Piura, Perú

2019

UNIVERSIDAD NACIONAL DE PIURA

FACULTAD DE INGENIERÍA INDUSTRIAL

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA

TESIS

"PROPONER UNA ARQUITECTURA DE SOFTWARE PARA EL DESARROLLO DE APLICACIONES EMPRESARIALES USANDO EL LENGUAJE DE PROGRAMACIÓN ESTRUCTURADO RPG EMPLEANDO ESTÁNDARES DE DISEÑO DE SOFTWARE."

LÍNEA DE INVESTIGACIÓN:

INFORMÁTICA, ELECTRÓNICA Y TELECOMUNICACIONES

SUBLÍNEA DE INVESTIGACIÓN:

COMPUTACIÓN

Tesista:

Br. NELSON ABEL BARRANZUELA IMAN

Asesor:

Ing. CARMEN LUCILA INFANTE SAAVEDRA, MSc.

Coasesor:

Dr. RIGO FÉLIX REQUENA FLORES

DECLARACION JURADA DE ORIGINALIDAD DE LA TESIS

Yo: **NELSON ABEL BARRANZUELA IMAN,** identificado con **DNI N° 42930505**, Bachiller de la Escuela Profesional de Ingenieria Informática, de la Facultad de Ingenieria Industrial y domiciliado en calle las Fresas MZ. N LT 8 AA.HH Los almendros del Distrito de Castilla, Provincia de Piura, Departamento de Piura, Celular: 968863159 Email: abelnelson27@gmail.com.

"Proponer una arquitectura de software para el desarrollo de aplicaciones empresariales usando el lenguaje de programación estructurado RPG empleando estándares de diseño de software."

DECLARO BAJO JURAMENTO: que la tesis que presento es original e inédita, no siendo copia parcial ni total de una tesis desarrollada, y/o realizada en el Perú o en el Extranjero, en caso contrario de resultar falsa la información que proporciono, me sujeto a los alcances de lo establecido en el Art. N° 411, del código Penal concordante con el Art.32° de la Ley N° 27444 y Ley del Procedimiento Administrativo General y las Normas Legales de Protección a los Derechos de Autor.

En fe de lo cual firmo la presente.

Piura, 06 de octubre del 2019

.....

DNI N° 42930205

Articulo 411.-El que, en un procedimiento administrativo, hace una falsa declaración en relación con hechos o circunstancias que le corresponde probar, violando la presunción de veracidad establecida por ley, será reprimido con pena privativa de libertad no menor de uno ni mayor de cuatro años.

Art. 4. Inciso 4.12 del Reglamento del Registro Nacional de Trabajos de Investigación para optar grados académicos y títulos profesionales –RENATI Resolución de Consejo Directivo N° 033-2016-SUNEDU/CD

UNIVERSIDAD NACIONAL DE PIURA
FACULTAD DE INGENIERÍA INDUSTRIAL
DECANATO

ACTA DE EVALUACIÓN Y SUSTENTACIÓN DE TESIS

Expediente N° 1478 / 2017

Los miembros del Jurado Calificador Ad-Hoc de la Sustentación de Tesis nombrado con Resolución N° **515-CF-FII-UNP-17** de fecha 28/08/2017 que suscriben, se reunieron en acto público en la sala de exposiciones de la Facultad de Ingeniería Industrial de la Universidad Nacional de Piura, el día **05 de Diciembre del 2019** a las **12:00 pm**, para evaluar la defensa de la Tesis titulada **"PROPONER UNA ARQUITECTURA DE SOFTWARE PARA EL DESARROLLO DE APLICACIONES EMPRESARIALES USANDO EL LENGUAJE DE PROGRAMACIÓN ESTRUCTURA RPG EMPLEADO ESTÁNDARES DE DISEÑO DE SOFTWARE"**, presentada por el Bachiller **NELSON ABEL BARRANZUELA IMÁN** y asesorado por la **MSc. CARMEN LUCILA INFANTE SAAVEDRA** y co-asesorado por el **Dr. RIGO FÉLIX REQUENA FLORES**.

Después de haber calificado el Informe Final de la Tesis, escuchada la sustentación y las respuestas a las preguntas formuladas por el Jurado, se le declara ...APROBADA.... para optar el Título de **INGENIERO INFORMÁTICO** con el puntaje de ...87... que corresponde al calificativo de ...SOBRESALIENTE

Jurado / Calificación	Presidente	Secretario	Vocal	Puntaje Promedio
Documento (Max 60 puntos)	50	50	50	50
Sustentación (Max 40 puntos)	37	37	37	37
PUNTAJE TOTAL	87	87	87	87

En consecuencia, el sustentante queda en condición de recibir el Título Profesional que se indica, conferido por el Consejo Universitario de la Universidad Nacional de Piura de conformidad con las Normas Estatutarias y la Ley Universitaria en vigencia.

Ciudad Universitaria, 05 de Diciembre del 2019

Dr. REUCHER CORREA MOROCHO	Dr. PEDRO ANTONIO CRIOLLO GONZALES	Dr. FRANCISCO JAVIER CRUZ VÍLCHEZ
PRESIDENTE	**SECRETARIO**	**VOCAL**

UNIVERSIDAD NACIONAL DE PIURA

FACULTAD DE INGENIERÍA INDUSTRIAL

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA

TESIS

"PROPONER UNA ARQUITECTURA DE SOFTWARE PARA EL DESARROLLO DE APLICACIONES EMPRESARIALES USANDO EL LENGUAJE DE PROGRAMACIÓN ESTRUCTURADO RPG EMPLEANDO ESTÁNDARES DE DISEÑO DE SOFTWARE."

Presidente: _____

Dr. REUCHER CORREA MOROCHO

Secretario: _____

Dr. PEDRO ANTONIO CRIOLLO GONZALES

Vocal: _____

Dr. FRANCISCO JAVIER CRUZ VILCHEZ

LÍNEA DE INVESTIGACIÓN:

INFORMÁTICA, ELECTRÓNICA Y TELECOMUNICACIONES

SUBLÍNEA DE INVESTIGACIÓN:

COMPUTACIÓN

DEDICATORIA

Dedico el presente trabajo a mis padres, hermanos y amigos por ser mi motivo de perseverancia y superación constante.

Por qué siempre me dieron su apoyo a lo largo de mi vida universitaria, brindándome los consejos a seguir y forjándome siempre por el camino del bien.

AGRADECIMIENTOS

Agradezco a mis asesores, de forma especial a la Ing. Carmen L. Infante Saavedra, MSc, que sin su apoyo el presente trabajo no hubiese sido posible, gracias por el apoyo incondicional que me brindaron en todo momento y sobretodo en el tiempo que conllevó realizar mi trabajo de investigación para optar el Título Profesional.

INDICE
1

INDICE DE FIGURAS 7

INDICE DE TABLAS 11

INDICE DE ANEXOS 12

Resumen 13

Abstract 14

Introducción 15

Capítulo 1: El problema de La investigación 1

1.1 Descripción y formulación del problema 1

1.2 Justificación de la investigación 2

1.3 Objetivos de la investigación 4

1.3.1 Objetivo general 4

1.3.2 Objetivos específicos 4

1.3.3 Delimitación de la investigación 5

Capítulo 2: Marco teórico 6

2.1 Bases teóricas científicas 6

2.1.1 ¿Qué es la arquitectura de software? 6

2.1.2 La arquitectura es un conjunto de estructuras 8

2.1.3 La arquitectura es una abstracción 9

2.1.4 Cada sistema software tiene una arquitectura 10

2.1.5 La arquitectura incluye comportamiento 10

2.1.6 No todas las arquitecturas son buenas 10

2.1.7 Patrones arquitectónicos 11

2.1.8 Catálogo de patrones arquitectónicos 14

2.1.8.1 Patrones modulares..... 14

2.2 Antecedentes..... 23

2.3 Glosario de Terminos..... 25

2.3.1 Análisis de la situación de la etapa de desarrollo de software..... 25

2.3.2 Desarrollo de ILE para el lenguaje de programación RPG IV..... 28

2.3.2.1 Entorno de desarrollo integrado ILE (por sus siglas en inglés)..... 28

2.3.2.2 Módulos..... 30

2.3.2.3 Programas de servicio..... 31

2.3.2.4 Enlazamiento (Binding): creación de programas *PGM y programas de servicio *SRVPGM 32

2.3.2.5 Directorios de enlace..... 33

2.3.2.6 Exportar programas de servicio 34

2.3.2.7 Lenguaje de enlazado 35

2.3.2.8 Grupos de activación 37

2.3.2.9 Comandos CL usados con ILE y RPG IV..... 39

2.4 Hipótesis..... 41

2.4.1 Hipótesis general 41

2.4.2 Hipótesis específica..... 41

Capítulo 3: Marco metodológico..... 42

3.1. Tipo y nivel de investigación 42

3.2. Modelo teórico..... 42

3.3. Diseño de investigación 43

3.4. Diseño de técnicas e instrumentos de recolección de datos e información 43

3.5. Tipo de técnicas de muestreo 43

3.6. Métodos, técnicas y uso de software de tratamiento de análisis de datos 44

Capítulo 4: Aplicación de la propuesta de la arquitectura de software 45

4.1 Requerimiento de usuario para el módulo de créditos. 45

4.2 Aplicación de la propuesta del marco de trabajo a empresa financiera 50

4.2.1 Programas del módulo desembolsos de Propuestas de créditos empresa financiera.52

4.2.2 Desarrollo del nuevo marco de trabajo para el módulo de Créditos de una empresa financiera. 76

4.2.2.1 Base de datos del requerimiento de check-list para el módulo de créditos de una empresa financiera. 77

4.2.2.2 Programa de servicio B07U0002: check-list propuesta de desembolso. 84

4.2.2.3 Programa de servicio B07U0002: utilidades para el manejo de Errores y ejecución de algunos comandos del sistema operativo. 91

4.2.2.4 Acoplando la nueva arquitectura de software a la existente. 96

Capítulo 5: Análisis y revisión de resultados finales 146

5.1 Reporte de resultados de la aplicación de marco de trabajo propuesto e interpretación 146

5.2 Discusión de resultados 156

Conclusiones 158

Recomendaciones 159

Referencias bibliograficas 160

Anexos 161

Anexo A: check-list para sistema de gestión de créditos para aprobación de desembolso 161

Anexo B: pase 162

Anexo C: pruebas 166

Anexo D: Código 193

Anexo E: código de fuente del módulo M01U0001 206

Anexo F: Código 214

INDICE DE FIGURAS

Figura 4.2.1.Diagrama de Arquitectura del nuevo Marco Propuesto en interacción con el existente……..…… 51

Figura 4.2.1.1. Subrutinas que componen el programa R7102310…… 55

Figura 4.2.1.2. Implementación de la subrutina sbrBusAlertCL del programa R7102310…….56

Figura 4.2.1.3.Vista De Diseño Pantalla P7102310 Del Programa R7102310…….. 58

Figura 4.2.1.4. Vista De Ejecución De Pantalla P7102310 Del Programa R7102310…….. 59

Figura 4.2.1.5. Diagrama del flujo de llamadas del programa R7102310……..60

Figura 4.2.1.6. Cambio introducido al programa R7102310 para el Check-List de Desembolso……62

Figura 4.2.1.7. Cambio introducido al programa R7102310 para el Check-List de Desembolso……..……..……..63

Figura 4.2.1.8. Fragilidad en el diseño del programa R7102310…….....64

Figura 4.2.1.9. Repetición innecesaria de algunas características del programa R7102310…….65

Figura 4.2.1.10. Opacidad del programa R7102310….....67

Figura 4.2.1.11. Ejecución del programa R7101504, consultas de propuestas de Desembolso……..... 68

Figura 4.2.1.12. Diagrama de componente del programa R7101504 y la interacción de sus subrtutinas……..70

Figura 4.2.1.13. Diagrama de componente del programa R7101504 y la interacción de sus subrtutinas……. 71

Figura 4.2.1.14. Rigidez en el programa R7101504……..73

Figura 4.2.1.15. Fragilidad del programa R7101504 en implementación de la subrutina sbrChkList…….…….. 74

Figura 4.2.1.16. Inmovilidad en el diseño del programa R7101504……....…….75

Figura 4.2.1.17. Opacidad del programa R7101504…….76

Figura 4.2.2.1.1.Modelo Entidad Relación para el requerimiento Check-List……79

Figura 4.2.2.2.1. Diagrama de componente del módulo M07U0002……85

Figura 4.2.2.2.2. Lenguaje de enlazado para el programa de servicio B07U0002……..87

Figura 4.2.2.2.3. Diagrama de componentes del programa de Servicio B07U0002……88

Figura 4.2.2.2.4.Resultado de aplicar el comando DSPSRVPGM al programa B07U0002.…….89

Figura 4.2.2.2.5.Resultado de aplicar el comando DSPSRVPGM al programa B07U0002, módulos que lo componen…… 90

Figura 4.2.2.2.6. Resultado de aplicar el comando DSPSRVPGM al programa B07U0002, subprocedimientos exportados……..90

Figura 4.2.2.2.7. Resultado de la ejecución del comando WRKBNDDIRE para el directorio de enlace SRVPBND…….. 91

Figura 4.2.2.3.1. Diagrama de Componente el módulo M01U0001……93

Figura 4.2.2.3.2. Lenguaje de enlazado para el programa de servicio B01U0001, miembro fuente: B01U0001.BND…….94

Figura 4.2.2.3.3. Programa de servicio B01U0001 instalado en el directorio de enlace SRVPBND……...95

Figura 4.2.2.3.4. Resultado de aplicar el comando DSPSRVPGM al programa B07U0001, módulo que lo componen……. 95

Figura 4.2.2.3.5. Resultado de aplicar el comando DSPSRVPGM al programa B01U0001, subprocedimientos exportados……. 96

Figura 4.2.2.4.1.Diagrama de interacción del programa R71M0018 con el programa R7102310 de arquitectura monolítica…… 101

Figura 4.2.2.4.2. Invocación del programa R71M0018 a través del lenguaje RPG desde el programa R7102310……...102

Figura 4.2.2.4.3. Lista de preguntas para el Check-List previo al desembolso de un crédito. ….103

Figura 4.2.2.4.4. Check-List previo al desembolso del crédito, registrar estado final de respuestas……104

Figura 4.2.2.4.5. Estructuras para los layouts de los registros de tabla de la aplicación Check-List…….. 106

Figura 4.2.2.4.6. Inclusión de archivo fuente R07U0002 a través de la directiva COPY en el módulo M07U0002…… 107

Figura 4.2.2.4.7. Procedimiento saveTipoExpedCheckList del módulo M07U0002 que utiliza estructuras de datos del archivo R07U0002…… 108

Figura 4.2.2.4.8. Procedimiento deleteTipoExpedCheckList del módulo M07U0002 que utiliza estructuras de datos del archivo R07U0002…… 109

Figura 4.2.2.4.9. Diseño de pantalla P71M0018, Check-List Previo al desembolso……111

Figura 4.2.2.4.10. Diseño pantalla P71M0018, Check-List Durante el desembolso de un crédito……..112

Figura 4.2.2.4.11. Pantalla P71M0018 en tiempo de ejecución, Check-List previo al desembolso……. 112

Figura 4.2.2.4.12. Pantalla P710018 en tiempo de ejecución, Check-List Durante el desembolso de un crédito…… 113

Figura 4.2.2.4.13. Controlador, Programa R71M0018, integración con la vista y el modelo de la aplicación Check-List…… 116

Figura 4.2.2.4.14. Estructuras de datos concretas creadas a partir de plantillas del archivo R07U008.RPGLE……...117

INDICE DE TABLAS

Tabla 5.1 Datos para el análisis estadístico de resultados……..102

Tabla 5.2 Respecto al efecto de los factores principales…….103

Tabla 5.3 Tabla de complejidad ciclomática de los marcos arquitectónicos analizados… 104

Tabla 5.4 Análisis de la varianza de la complejidad cíclica de una aplicación empleando el nuevo marco de trabajo…… 105

Tabla 5.5 Comparación de arquitecturas : Monolítica y Modular……106

INDICE DE ANEXOS

Anexo A: Check-List para Sistema de Gestión de créditos para aprobación de desembolso 115

Anexo B : Pase a producción de la Aplicación Check-List …….. 116

Anexo C: Pruebas Unitarias Enviadas al Área de Calidad……120

Anexo D : Código de Fuente del módulo M07U0002……..147

Anexo E : Código de Fuente del Programa R71M0018……. 168

Resumen

El presente proyecto tiene como objetivo proponer una arquitectura de software para el desarrollo de aplicaciones empresariales utilizando el lenguaje de programación estructurado RPG IV o también denominado simplemente RPG, con el fin de mejorar el proceso de desarrollo de software aplicando modernas técnicas de arquitectura de software, mejorando significativamente el cicclo de desarrollo, diseño y mantenimiento de la aplicación; resolviendo problemas como : duplicar código fuente, desencadenar o introducir nuevos errores en el código fuente, reducir la cantidad de tiempo para detectar errores.

Para la realización del presente proyecto se ha tomado como base las nuevas técnicas de desarrollo de software que la industria ofrece, así como el nuevo modelo de programación del lenguaje RPG IV denominado ILE.

En el desarrollo de la aplicación software sobre la cual se aplica la arquitectura de software propuesta se hizo uso de la metodología RUP (Rational Unified Process), conocida por ser iterativa e incremental. Se identificaron los procesos relacionados al área de gestión de créditos de una empresa financiera, posteriormente se determinaron los requerimientos de una nueva aplicación modelando los procesos y construyendo los diagramas. Así mismo se diseñaron las interfaces del sistema.

En la implementación se codificó el software en RPG IV, se implementó la base de datos a través de DB2 y se hicieron las pruebas de funcionamiento respectivas.

Palabras Clave: Mejorar diseño de software / Arquitectura de software / Patrones Arquitectónicos.

Abstract

The objective of this project is to implement an software architecture for the process of software development for "Enterprise Applications" in order to streamline the processes, the treatment and the integrity of the data, which allows maintaining a database updated, significantly improving the quality of the process of development and design in the lifecycle of the software, solving the problems that arose as: duplication of code, triggering new bugs in the source code, too much amount of time for detecting errors.

For the realization of this project, it has been based on the new and modern development software techniques offered by the industry, as well as the rpg IV's new programing model called ILE.

In the development of the software application on which the proposed software architecture is implemented, the RUP (Rational Unified Process) methodology, known as iterative and incremental, was used. First, the processes related to the admission area were identified, then the requirements of the system were determined through the modeling of processes and the construction of the diagrams. Likewise, the system interfaces were designed.

In the implementation, the software was coded in RPG IV, the database was implemented through DB2 and the respective performance tests were carried out.

Keywords: Improve software design / software architecture / Patterns of software architecture.

Introducción

El uso del lenguaje es sin duda una de las herramientas más poderosas que ha construido el hombre para poder hacer una abstracción del mundo que le rodea, construir su realidad, y poder conservar sus vivencias, recuerdos y todas cosas que le son de utilidad. Es así como el uso de este le ha permitido lograr grandes avances científicos que le han sido indispensables para su supervivencia a lo largo de la historia, pero el lenguaje no es más que una abstracción y su uso está supeditado a ciertas normas preestablecidas por las personas que las usan, o que han decidido llegar a un acuerdo mutuo acerca de su uso. Por lo tanto, se diría que es una mera convención a la cual estamos ligados todos los seres humanos, esto sin dudas genera muchas implicancias que están lejos del alcance de este trabajo; muchas de ellas filosóficas. Pero siendo el lenguaje una abstracción de la mente humana para poder comunicarnos entre nosotros, este mismo proceso ha sido usado para construir los lenguajes que permite ejecutar un conjunto de órdenes en el computador a través de sentencias que luego son interpretadas por este, a estos lenguajes se les denomina lenguajes de programación. Por lo tanto, haciendo uso de estos a un grado de nivel abstracción más alto, podemos crear las modernas aplicaciones software que usamos en nuestros días, pero a medida que evolucionan, surge la necesidad de poder hacer uso correcto de su aplicación, y ha llegado a ser una tarea que ha generado nuevas ramas en la ingeniería del software, apareciendo lo que hoy se llama, Arquitectura de Software.

En el siglo XVI el gran filósofo francés René Descartes publicó su gran obra filosófica denominada, “El Discurso del Método” y con ello da inicio al renacimiento, y una nueva era para la razón moderna, en la segunda parte de dicho discurso

menciona lo siguiente: "…No existe tanta perfección en obras compuestas de muchos elementos y realizadas por diversos maestros como en aquellas ejecutas por uno solo. Así vemos que los edificios que un solo arquitecto ha comenzado y acabado suelen ser más bellos y mejor ordenados que aquellos otros que varios han tratado de reformar." **(Descartes, 1637, pág. 16)** Podemos notar aquí la gran observación que hace el gran filósofo para describir los resultados que se obtienen cuando no se usan los elementos de construcción armoniosamente o cuando se hace una construcción sin tener un plano a seguir. Diríamos que los ladrillos de un lenguaje de programación son los elementos que posee para poder realizar un programa de computadora. Es por ello por lo que a lo largo del presente trabajo se hará un estudio de las nuevas técnicas para desarrollo de software con la finalidad de poder aplicarlas usando el lenguaje de programación RPG IV, se va a diseñar una arquitectura de software para luego proponer su aplicación en algunas empresas que utilizan dicha tecnología. Se va a analizar la situación actual de las técnicas que se usan para su construcción y dar un diagnóstico de los problemas que surgen al realizar mantenimiento de estas, ya sea para agregar nuevas funcionalidades o para soportar nuevos requisitos de negocio.

Capítulo 1: El problema de La investigación

1.1 Descripción y formulación del problema

En la actualidad existen empresas que poseen software que fue escrito hace muchos años, especialmente las empresas dedicadas a finanzas, es así como se les denomina aplicaciones heredadas o "legacy applications", uno de los lenguajes más usados para construir este tipo de aplicaciones es RPG IV o simplemente RPG, un lenguaje de programación estructurado desarrollado por la empresa IBM. Uno de los factores fundamentales para usar este tipo de aplicaciones es su rapidez de ejecución en operaciones transaccionales con grandes volúmenes de información y alta demanda de peticiones. Pero la característica fundamental de este tipo de aplicaciones es su diseño, muchas fueron construidas usando técnicas empíricas de programación y muchas de las personas que las desarrollaron no tenían ningún concepto de lo que es hacer un diseño de software o lo que es arquitectura de software.

Pero lo más sorprendente es que aún hoy las personas que hacen mantenimiento a dichas aplicaciones o que desarrollan una nueva usando la misma tecnología, cometen los mismos errores de diseño y no aplican ningún concepto de lo que es arquitectura de software. Esto se debe a que muchos de ellos no conocen los conceptos fundamentales del lenguaje de programación RPG IV y sus elementos, nunca se cuestionan o preguntan si existe una mejor manera; si es que es posible hablar de ello, de realizar un diseño arquitectónico elegante y construir un marco de trabajo que sirva para aplicar las nuevas técnicas de programación que se han desarrollado en los últimos tiempos, pues como se sabe la industria del software es bastante joven.

El problema aparece; como siempre suele suceder, de la falta de una comprensión profunda de las herramientas que se usan, lo cual lleva a cometer muchos errores durante

la fase de diseño y desarrollo. También debido a la naturaleza misma del lenguaje de programación, pues, en primer lugar, este es uno de los más antiguos de la industria y quizá uno de los más difíciles de aprender, tampoco se enseña en ambientes académicos pues es una tecnología privada, pero por qué elegir una tecnología privada objetaran algunos, esa respuesta está fuera del alcance del presente trabajo. Sin embargo, podría decirse en su defensa que ambos tipos de software, sea libre o privativo, han progresado muy de la mano.

Es necesario que las empresas progresen a medida que la tecnología avanza, aplicando las técnicas que la industria del software ha desarrollado, alineen estos cambios con los objetivos de negocio que la empresa persigue, de lo contrario no podrá evolucionar según los cambios que el mundo le exige; hoy se pueden ver los grandes esfuerzos que se realizan para poder adaptar las aplicaciones existentes a los cambios que se requieren para soportar nuevas necesidades de negocio, si no se toman en cuenta las innovaciones que se tienen hoy en día, las empresas no podrán crecer y competir con las empresas de su entorno.

1.2 Justificación de la investigación

El desarrollo de una aplicación software es una tarea que necesita del conocimiento de técnicas que proporciona la ingeniería del Software para lograrla. Pero, ésta tarea en sus inicios fue desarrollada de forma empírica y no se usó ninguna técnica de ingeniería para esta labor, uno de los principales motivos, es como se sabe, la ingeniería de software es una de las ramas de saber humana más jóvenes, y en este corto plazo de su desarrollo las disciplinas que han surgido con su advenimiento son muchas, en entre ellas es la arquitectura de software, que a su vez está dentro de lo que es el diseño de software. Los lenguajes de programación modernos, por ejemplo, Java, C#, C++, JAVA SCRIPT, entre

otros, proporcionan uno de los conceptos fundamentales al momento de desarrollar una aplicación software; esta herramienta o concepto, es el paradigma de programación orientado a objetos, con dicha herramienta se puede modelar las necesidades de negocio a resolver de una forma más elegante haciendo uso de todas las herramientas que la ingeniería de software proporciona. Pero, existen otras tecnologías que no proporcionan o no cuentan con la característica de la programación orientada a objetos, esto representa un gran inconveniente al momento de diseñar e implementar un sistema software para muchos programadores jóvenes. Una de estas tecnologías para desarrollar una aplicación software es el lenguaje de programación estructurado RPG, y se resalta estructurado, porque es un gran reto poder desarrollar una aplicación usando todas las herramientas y las buenas prácticas que nos proporciona la ingeniería del software.

En la actualidad existen muchas aplicaciones que están construidas con el lenguaje de programación RPG, pero construidas pobremente o han usado un mal diseño lo cual genera muchos problemas a la hora de darles de mantenimiento, a esto se suma del poco o ningún conocimiento por parte de las personas que realizan dichas tareas, muchas de ellas son personas que están a punto de retirarse laboralmente, lo cual representa una gran problema a la hora de lidiar con ellas, pues no quieren aprender nada nuevo ya, existen sus excepciones, pero esto no es suficiente. Otro problema que se da es cuando se va a desarrollar una nueva aplicación, los conceptos y herramientas que proporciona el lenguaje de programación RPG no son conocidos por las personas que los usan, incluso algunos programadores jóvenes que trabajan con esta las desconocen, esto genera grandes problemas y resulta en un pobre diseño, lo cual afecta en gran medida las tareas de mantenimiento.

Uno de los grandes problemas que generan todo lo antes mencionado es relacionado en las tareas de mantenimiento, ya sean éstas arreglar un error de sistema o agregar una nueva

característica, pues cuando se resuelva un error, este genera otros, esto a su vez causa la molestia de los usuarios del sistema, en lo relacionado a agregar una nueva funcionalidad el problema surge debido a la gran dificultad que con lleva realizarla, por no decir que es una tarea casi imposible. El tiempo que toma que toma encontrar el error que ha ocurrido en el sistema es muy alto, en algunos casos esto causa pérdidas a la empresa.

Por ello, el presente trabajo pretende aplicar las nuevas técnicas de ingeniería de software al lenguaje de programación estructurado RPG, y generar un marco de trabajo para mejorar el desarrollo de sistemas software usando esta tecnología.

1.3 Objetivos de la investigación

1.3.1 Objetivo general

Proponer una arquitectura de software para el desarrollo de aplicaciones empresariales usando el lenguaje de programación estructurado RPG empleando estándares de diseño de software.

1.3.2 Objetivos específicos

- Aplicar las nuevas técnicas de Ingeniería y arquitectura de software usando el lenguaje de programación estructurado RPG IV
- Maximizar la reutilización del código fuente en la fase de desarrollo de la aplicación
- Evaluar el impacto de un marco de trabajo para el desarrollo de aplicaciones empresariales usando RPG IV

- Incrementar la productividad de los equipos de desarrollo durante la etapa de desarrollo y de mantenimiento

1.3.3 Delimitación de la investigación

Las áreas usuarias de las aplicaciones, el equipo programadores que se encargan de su creación y mantenimiento, otros desarrolladores que nunca han usado RPG como lenguaje de desarrollo formaran parte de la unidad a investigar

Capítulo 2: Marco teórico

2.1 Bases teóricas científicas

2.1.1 ¿Qué es la arquitectura de software?

El termino o palabra "Arquitectura de Software", es un término muy difícil de definir, ya que muchas personas involucradas en el mundo del desarrollo de software han tratado de darle un concepto, es por ello por lo que se va a recurrir a distintas fuentes para poder dar una perspectiva de todo el avance que se ha venido dando respecto al estudio de esta apasionante materia de estudio. Las definiciones que se van a citar serán varias, por lo cual al final de este apartado se tomará una de ellas de acuerdo con el contexto en el cual se ha desarrollado el presente trabajo.

Ian Gorton en su famoso libro denominado "Green Book" señala que definir el término "arquitectura de software" es una actividad potencialmente peligrosa. Realmente no hay definición ampliamente aceptada por la industria. **(Gorton, 2011, p. 2)**

Empecemos con la definición que se considera la más ampliamente aceptada por muchos otros autores, esto ya que de acuerdo con el trabajo de investigación realizado se ha visto que tomada como referencia en las diferentes fuentes de información recopiladas. "La arquitectura de software de un programa o sistema de cómputo, es el conjunto de estructuras necesarias para entender el sistema, las cuales comprenden los elementos software, las relaciones entre ellos y sus propiedades visibles". **(Bass, Clements, & Kazman, 2013, p. 4)**

Una de las principales organizaciones de ingenieros alrededor del mundo, el IEEE; Instituto de Ingeniería Eléctrica y Electrónica por sus siglas en inglés, proporciona en

su documento de estándares ,“IEEE Recommended Practice for Architectural Description of Software-Intensive Systems”, una definición para la arquitectura de software : “La organización fundamental de un sistema representada en sus componentes, sus relaciones entre sí, y al entorno, y los principios que guían su diseño y evolución”. **(The Institute of Electrical and Electronics Engineers, Inc., 2000, p. 3)**

Por otra parte, uno de los trabajos que han aportado grandes luces al campo de la Arquitectura de Software es el desarrollado por Martin Fowler, en uno de ellos nos da las siguientes definiciones: “En los proyectos software más exitosos, los desarrolladores expertos trabajando en tal o cual proyecto tienen un entendimiento compartido del diseño del sistema. Este entendimiento es llamado arquitectura. Este entendimiento incluye cómo el sistema es dividido en componentes y cómo estos interactúan a través de sus interfaces. Estos componentes están usualmente compuestos de componentes más pequeños, pero la arquitectura sólo incluye los componentes e interfaces que son entendidos por todos los desarrolladores” **(Fowler, 2002, págs. 2-3)**, esta definición está en más sintonía con las aplicaciones así llamadas “Legacy Applications”, ya que la arquitectura no existe sino en la cabeza de los desarrolladores que le dan soporte.

Ahora tomaremos una definición proporcionado en el mismo trabajo de Martin Fowler que nos permitirá adecuarla en el contexto que se desarrolla el presente trabajo, se define la Arquitectura de Software como: “La Arquitectura de Software es el conjunto de decisiones que se desean tomar anticipadamente en un proyecto, pero que no son probadamente correctas más que cualquier otra.” **(Fowler, 2002, pág. 3)**

Es en esta última definición en la cual Martin Fowler considera al lenguaje de programación un componente arquitectónico, pues es una decisión que se toma en las primeras etapas del desarrollo del proyecto, es por lo tanto una restricción técnica a la que se está supeditado, en este caso se estará trabajando con el lenguaje de programación estructurado RPGLE, para explicar más a detalle, el autor considera que un componente arquitectónico es aquel que es un muy importante para el diseño y desarrollo del proyecto, por lo que si dicho componente se puede retirar del diseño sin afectarlo, entonces este deja de ser arquitectónico, podemos citar a las bases de datos, en muchos casos esta se puede reemplazar por cualquier otra sin poder afectar al diseño del proyecto, por lo tanto deja de ser un componte arquitectónico, pero basado nuevamente en la definición proporcionada por Martin Fowler, en este caso la base de datos también es un componente arquitectónico del proyecto ya que se está sujeto a usar DB2.

2.1.2 La arquitectura es un conjunto de estructuras

Una estructura es simplemente un conjunto de elementos unidos por una relación. Los sistemas software están compuestos de muchas estructuras, y no ninguna de ellas puede decirse que es la arquitectura por si sola. Hay tres categorías de estructuras las cuales desempeñan un rol importante en el diseño, documentación y análisis de arquitecturas.

- **Módulos:** Los módulos son asignados a responsabilidades específicas de computación y son las bases de asignación de trabajo para los equipos de programación, por ejemplo: el equipo A trabaja en la bases de datos, el equipo B trabaja en las reglas de negocio, etc. Existe otra clase de módulo

que resulta del análisis y diseño orientado a objetos; este es el diagrama de clases. Las estructuras módulos son estructuras estáticas.

- **Componente y conector:** Son estructuras dinámicas, están enfocadas en la manera cómo los elementos interactúan entre ellos en tiempo de ejecución para llevar a cabo las funciones del sistema. El conector describe cómo los componentes interactúan, por ejemplo: Objetos, threads, o procesos, los sockets, middleware como CORBA o memoria compartida.
- **Estructuras de asignación:** Esta tercera clase de estructura describe el mapeo de las estructuras de software a los entornos organizacional, de desarrollo, instalación y ejecución del sistema y la forma cómo se comunican estos a la red y/o base de datos. Por ejemplo, los módulos son asignados a los equipos de desarrollo, y estos asignados a lugares en la estructura de archivos para la implementación, integración y despliegue. Los componentes son desplegados sobre el hardware para ser ejecutados.

2.1.3 La arquitectura es una abstracción

La arquitectura de software es ante todo una abstracción de un sistema que selecciona ciertos detalles y suprime otros. En todos los sistemas modernos, los elementos interactúan con otros a través de interfaces que dividen los elementos en públicos y privados. La arquitectura tiene que ver con el lado público de esta división, los detalles privados de estos elementos no se consideran arquitectónicos. Aunque yendo más allá de las interfaces, la abstracción arquitectónica nos permite mirar al sistema en términos de sus elementos, cómo están organizados, cómo interactúan, cómo están compuestos, cuáles son las propiedades que nos permiten entender el sistema, etc. Esta abstracción es esencial para poder comprender la complejidad de un sistema. **(Bass, Clements, & Kazman, 2013, p. 6)**

2.1.4 Cada sistema software tiene una arquitectura

Aún cuando cada sistema tiene una arquitectura, esto no necesariamente implica que la arquitectura es conocida por alguien. Pues puede ocurrir que la gente que diseño el sistema ha dejado la empresa hace mucho tiempo, la documentación ha desaparecido y todo lo que se tiene es el código binario ejecutable. Esto revela la diferencia entre la arquitectura de un sistema y la representación de dicha arquitectura. En el caso más trivial, un sistema en sí mismo es un solo elemento, un elemento nada interesante y probablemente inútil, pero una arquitectura, al fin y al cabo. **(Bass, Clements, & Kazman, 2013, p. 6)**

2.1.5 La arquitectura incluye comportamiento

El comportamiento de cada elemento es parte de la arquitectura siempre y cuando dicho comportamiento puede ser usado para comprender el sistema. Este comportamiento plasma cómo los elementos interactúan entre ellos, lo cual claramente es la definición de arquitectura de software. No todos los comportamientos deben ser documentados en la arquitectura, puesto que existen aquellos que no son importantes y sólo conciernen al nivel arquitecto del sistema. **(Bass, Clements, & Kazman, 2013, p. 7)**

2.1.6 No todas las arquitecturas son buenas

La definición es indiferente ya sea que la arquitectura para un sistema es una buena o es una mala. Una arquitectura podría permitir o imposibilitar el éxito de del comportamiento, atributos de calidad, y los requerimientos del ciclo de vida de un sistema. Se puede dar el caso que por restricciones se elija una arquitectura al azar la cual no se ha probado y que el resultado final no cumpla con los requisitos que se

esperaban, esto se genera el fracaso total del proyecto. **(Bass, Clements, & Kazman, 2013, p. 6)**

2.1.7 Patrones arquitectónicos

Antes de entrar con detalle a definir lo qué es un patrón arquitectónico, se va a explicar con una reflexión los detalles que existen detrás de esta abstracción, y se enfatiza abstracción por qué todo el conocimiento que el hombre ha adquirido a lo largo de su existencia en la tierra, ha sido a través de representar la naturaleza que le rodea con símbolos que le permiten dotar de significado a los objetos que observa a través de su experiencia o para definir las cosas abstractas como la fe y el amor, se puede poner como claro ejemplo el alfabeto con el que he escrito éstas líneas, cada símbolo representa el sonido que usamos para plasmar nuestras ideas. Menciono todo esto, por qué quiero alentarlos a reflexionar y analizar el desarrollo que hacemos diariamente, es parecido a cuando contemplamos la naturaleza, toda ella está llena de misterios a la que todo hombre debería abrir su mente para absortarse en sus pensamientos, mirarla como hermoso cuadro de pintura hecha por cualquiera de los grandes maestros; Picasso, Rafael, da Vinci, Delacroix, Fragonard, Botticelli, etc. por mencionar algunos, una infinita biblioteca a la debemos mirar con los ojos de un niño, es decir pensar por nosotros mismos, dejar las convenciones y seamos nuevamente curiosos para poder abarcar las cosas desde un punto de vista más allá de nuestro pensamiento lineal.

Es así como ella nos muestra diferentes formas en toda su maravillosa vastedad, infinitas por no decir otra cosa, pero que, en muchas de ellas, al ser contempladas por nuestros sentidos, podemos notar formaciones a las que hemos definidos como patrones, fijémonos nada más en los girasoles o en las galaxias que conforman nuestro

universo, en los caparazones de las tortugas. Usamos entonces patrón para definir aquello que es semejante entre las cosas.

Para finalizar, voy a dejar una cita del mejor pensador del siglo XX que ha podido tener la humanidad, me refiero a Albert Einstein: "La más profunda y maravillosa experiencia que puede tener un hombre es el sentido del misterio. Es el principio que yace bajo la religión, las artes y la ciencia. Aquel que nunca haya tenido esta experiencia me parece, sino muerto, al menos ciego"**.**

Como se mencionó en los párrafos anteriores, la experiencia es la que muestra al hombre los misterios de cada cosa con la que experimenta, y tal como sostienen Bass, Clements y Kazman lo mismo ocurre con los patrones arquitectónicos, ellos son descubierto en la práctica, uno no los inventa, uno los descubre. Catalogar patrones es similar a el trabajo de un botánico o zoólogo: Describir patrones y describir sus características compartidas. Y tal como el botánico, zoólogo, o ecologista, el catalogador de patrones se empeña en entender las características llevan a diferentes comportamientos y diferentes respuestas a las condiciones del entorno. Por esta razón nunca habrá una lista completa de patrones: los patrones emergen espontáneamente en reacción a las condiciones del entorno, siempre que esas condiciones cambien, nuevos patrones emergerán. **(Bass, Clements, & Kazman, 2013, p. 203)**

Resumiendo, las ideas de los autores citados, ellos plantean los siguientes puntos acerca de la os patrones arquitectónicos:

- Es un paquete de decisiones de diseño que es encontrado repetidamente en la práctica.
- Tiene propiedades conocidas que permiten reusabilidad.
- Describe una clase de arquitectura.

A continuación, se detalla la forma en que un patrón arquitectónico puede establecer las relaciones que lo gobiernan de acuerdo con Bass, Clements y Kazman, entonces se dice que estos establecen relación entre:

- Contexto: Una situación recurrente, común en el mundo que da lugar a un problema.
- Un problema: El problema apropiadamente generalizado, que aparece en el contexto dado. La descripción del patrón esboza el problema y sus variantes y describe cualquier complementariedad o fuerzas que se le oponen.
- Una solución: Una solución arquitectural exitosa al problema, apropiadamente abstraída. La solución describe las estructuras arquitectónicas que resuelven el problema, incluyendo cómo balancear las distintas fuerzas en el trabajo. La solución describe las relaciones estáticas entre los elementos que conforma la solución arquitectónica o describirá su comportamiento en tiempo de ejecución. La solución para un patrón es determinada y descrita por:
- Un conjunto de tipos de elementos, estos pueden ser: repositorios de datos, procesos y objetos.
- Un conjunto de mecanismos de interacción o conectores, por ejemplo: llamadas a métodos, eventos o mensajes de bus de datos.
- Un diseño topológico de componentes.
- Un conjunto de restricciones protegiendo la topología, y los mecanismos de interacción.

2.1.8 Catálogo de patrones arquitectónicos

A continuación, se va a listar una variedad de patrones arquitectónicos de utilidad y que son ampliamente usados. De acuerdo con Bass, Clements y Kazman estos patrones se pueden agrupar de la siguiente manera: patrones de elementos de tiempo de ejecución (runtime elements) y patrones con elementos para tiempo de diseño. Para cada patrón se va a listar el contexto, problema y solución. **(Bass, Clements, & Kazman, 2013, p. 205)**

Además, ellos alegan que los patrones se pueden clasificar por el tipo dominante de elementos que ellos muestran: patrones modulares muestran módulos, patrones de componentes y conectores muestran componentes y conectores, y los patrones de asignación muestran una combinación de elementos softwares y no elementos software. **(Bass, Clements, & Kazman, 2013, p. 205)**

2.1.8.1 Patrones modulares

2.8.1.1 Patrón de capas

- **Contexto:** Todo sistema complejo experimenta la necesidad de desarrollar y evolucionar ciertas partes del sistema independientemente. Por esta razón los desarrolladores necesitan una clara y bien documentada separación de responsabilidades de forma tal que los módulos puedan ser mantenidos y desarrollados de forma independiente.
- **Problema:** El software necesita ser segmentado de forma tal que los módulos puedan ser desarrollados y evolucionar separadamente con poca interacción entre las partes, soportando portabilidad, susceptible de modificación, y reusabilidad.

- **Solución:** Para lograr la separación de responsabilidades el patrón de capas divide el software en unidades denominadas capas. Cada capa es una agrupación de módulos que ofrecen un conjunto cohesivo de servicios. Hay restricciones de tipo de "allowed-to-use"; es decir, autorizado-a-usar entre las capas: la relación debe de ser unidireccional. Las capas particionan completamente un conjunto del software, y cada partición es expuesta a través de una interfaz pública.

2.8.1.2 Patrón modelo vista controlador

- **Contexto:** El software de interfaz de usuario es típicamente la más frecuente porción modificada de una aplicación interactiva. Por esta razón es importante mantener las modificaciones al software de interfaz de usuario separada del resto del sistema.
- **Problema:** Cómo la funcionalidad de interfaz de usuario puede ser mantenida separada de la funcionalidad de la aplicación y aun así ser receptiva a los datos de entrada de los usuarios o cambios a los datos subyacentes de la aplicación.
- **Solución:** El patrón modelo vista controlador separa la funcionalidad de la aplicación en tres clases de componentes.
    - **Un modelo,** el cual contiene los datos de la aplicación
    - **Una vista,** la cual muestra o visualiza algunos datos subyacentes e interactúa con los usuarios.
    - **Un controlador:** El cual media entre el modelo y la vista y maneja las notificaciones de los cambios de estado.

2.8.1.3 Patrón pipe y filter

- **Contexto:** Muchos sistemas son requeridos para transformar flujos discretos de información, desde la entrada a la salida. Muchos tipos de transformaciones ocurren en la practica así es que es deseable crear estas como partes independientes y partes reusables.
- **Problema:** Tales sistemas necesitan ser divididos en componentes reusables de bajo acoplamiento con mecanismos de interacción simples y genéricos. De este modo ellos pueden ser flexiblemente combinados entre ellos.
- **Solución:** El patrón de interacción en el patrón pipe-and-filter es caracterizado por sucesivas transformaciones de flujos de información. Los datos de información llega a la entra de la puerta del filtro, son transformados, y luego son pasados a través de su puerto de salida a través de una tubería al siguiente filtro.

2.8.1.4 Patrón cliente servidor

- **Contexto:** Existen recursos compartidos y servicios que una gran cantidad de clientes distribuidos desean acceder, y por el cual deseamos controlar el acceso o la calidad del servicio.
- **Problema:** Manejando un conjunto de recursos y servicios compartidos, se puede promover la modificabilidad y reusabilidad, excluyendo servicios comunes y teniéndolos que modificar en una sola localización o un pequeño número de localizaciones. Se desea mejorar la escalabilidad y disponibilidad centralizando el control de estos recursos y servicios.
- **Solución:** Los clientes interactúan solicitando servicios de los servidores, los cuales proveen un conjunto de servicios. Algunos

componentes pueden actuar como cliente y servidor. Podría haber un servidor central o multiples servidores distribuidos.

2.8.1.5 Patrón peer-to-peer

- **Contexto:** Entidades computaciones distribuidas; cada una de las cuales es considerada igualmente importante en términos de inicialización e interacción y cada una de las cuales provee sus propios recursos, necesitan cooperar y colaborar para proveer un servicio a una comunidad de usuarios distribuidos.
- **Problema:** Cómo puede un conjunto de entidades computaciones iguales puede ser conectada entre sí a través de un protocolo común de formar que puedan ser organizadas y compartir sus servicios con alta disponibilidad y escalabilidad.
- **Solución:** En el patrón Peer-to-Peer los componentes interactúan directamente como pares. Todos los pares son iguales y ningún par o grupo de pares puede ser critico para la salud del sistema. La comunicación Peer-to-Peer es típicamente una interacción solicitud/respuesta sin la asimetría encontrada en el patrón cliente servidor. Cada par provee y consume servicios similares y usa el mismo protocolo. Los conectores en los sistemas Peer-to-Peer involucran interacciones bidireccionales reflejando la comunicación de dos caminos que puede existir entre dos o más pares.

2.8.1.6 Patrón arquitectónico orientado a servicios

- **Contexto:** Un número de servicios son ofrecidos y descritos por un proveedor de servicios y consumidos por consumidores de servicios. Los

consumidores del servicio necesitan ser capaces de entender y usar esos servicios sin ningún conocimiento detallado de su implementación.

- **Problema:** Cómo se puede soportar interoperabilidad de componentes distribuidos ejecutándose en diferentes plataformas.
- **Solución:** El patrón arquitectónico orientado a servicio (SOA) describe una colección de componentes distribuidos que provee y/o consume servicios. En una arquitectura SOA los componentes proveedores de servicios y los componentes consumidores de servicios pueden ser implementados en diferentes lenguajes y plataformas. Los servicios ampliamente independientes, y a menudo pertenecen a diferentes sistemas o incluso organizaciones.

2.8.1.7 Patrón publish-subscribe

- **Contexto:** Hay un número independiente de productores y consumidores de información que deben interactuar. El número preciso y naturaleza de los productores y consumidores de información no es predeterminado o fijo, tampoco el tipo de información que ellos comparten.
- **Problema:** Cómo se puede crear un mecanismo de integración que soporte la habilidad de transmitir mensajes entre los productores y los consumidores de forma tal que ellos estén de la identidad de ellos entre sí o potencialmente incluso de su existencia.
- **Solución:** En el patrón Publish-Subscribe los componentes interactúan a través del anuncio de mensajes, o eventos. Los componentes pueden suscribirse a conjunto de eventos. Es trabajo de la infraestructura del patrón Publish-Subscribe en tiempo de ejecución asegurar que cada

evento publicado es entregado o llevado a todos los suscritores de dicho evento.

2.8.1.8 Patrón de datos compartidos (shared-data)

- **Contexto:** Varios componentes computacionales necesitan compatir y manipular gran cantidad de información. La información no solo pertenece a uno de aquellos componentes.
- **Problema:** Cómo los sistemas pueden manipular y almacenar la data almacenada la cual es accedida por múltiples componentes independientes.
- **Solución:** En el patrón Shared-Data, la interacción es dominada por el persistente intercambio de información entre múltiples consumidores de información y al menos un almacén de datos compartidos. El intercambio puede ser por los componentes que acceden a la información o por el almacén de datos. El tipo de conector es un componente que lee y escribe información. El modelo general computacional asociado los sistemas que comparten data es que los componentes que acceden a ella ejecutan operaciones que requieren data del almacén de datos y escribir los resultados a uno o más almacenes de datos.

2.1.8.2 Patrones componente y conector

2.1.8.2.1 Patrón broker

- **Contexto:** Muchos sistemas son construidos a partir de una colección de servicios distribuidos a través de múltiples servidores. Implementar estos sistemas es complejo por qué se necesita preocupar acerca cómo los sistemas interoperaran. Cómo se conectarán el uno al otro y cómo van a

intercambiar información, así cómo también mantendrán la disponibilidad del servicio.

- **Problema:** Cómo estructurar un software distribuido de forma tal que los usuarios de los servicios no necesitan saber nada acerca de la naturaleza y localización de los proveedores de los servicios, facilitando el cambio dinámico del enlace entre usuarios y proveedores.
- **Solución:** El patrón de los servicios separa los usuarios de los servicios (clientes) de los proveedores de servicios (servidores) insertando un intermediario, llamado un bróker. Cuando un cliente necesita un servicio este solicita un bróker a través de un servicio de interfaz. El bróker entonces envía la solicitud del cliente a el servidor, el cual procesa la solicitud. El resultado del servicio es enviado de vuelta al bróker, el cual entonces retorna el resultado (y cualquier excepción) al cliente solicitante. De esta forma el cliente permanece completamente ignorante de la identidad, localización, y características del servidor.

2.1.8.3 Patrones de asignación o allocation patterns

2.1.8.3.1 Patrón map-reduce

- **Contexto:** Los negocios tienen una apremiante necesidad de analizar rápidamente enormes volúmenes de data que ellos generan o acceden, a escala de petabytes. Ejemplos incluyen logs de interacción en una red social, documentos o repositorios de data, y pares de enlaces web <source, target> de un motor de búsqueda.
- **Problema:** Para muchas aplicaciones con conjuntos de información ultra largos, ordenar la data y luego analizar la data agrupada es suficiente. Como se puede ver, el patrón Map-Reduce es desarrollar un medio que

resuelva el problema de ordenar un gran volumen de información de forma eficiente y en paralelo y de esta forma proveer al programador un medio que le permita realizar el análisis que el desee con la data que se ha ordenado y agrupado.

- **Solución:** El patrón Map-Reduce plantea una solución que necesita de tres partes. La primera de ella es una infraestructura especializada que se ocupe de asignar el software a los nodos hardware en un entorno de computación masivamente paralelo y maneje el ordenamiento de la información a medida que esta es necesitada. Un nodo puede ser un procesador a medida o un chip multiprocesador. Como segundo y tercera parte se tiene los dos programas llamados "Map" y "Reduce" que implementan las funciones de ordenar y reducir la información.

  La función Map toma como entrada una llave (key1) y un conjunto de datos. Su propósito es filtrar y ordenar el conjunto de datos. Todo el análisis pesado es llevado a cabo en la función Reduce. Una segunda clave(key2) también es importante en la función Map. Esta llave es para ordenar el conjunto de datos. La salida de la función Map consiste en un par <key2,valor>, donde key2 es el valor ordenado y valor es derivado del registro de entrada.

2.1.8.3.2 Patrón multicapa

De acuerdo con Ben, Clements y Kazman el patrón arquitectónico puede clasificarse como un patrón modular o un patrón de asignación (Allocation Pattern), pero esto depende del criterio para definir las capas. Las capas pueden ser creadas para agrupar funcionalidades similares del software, en tanto que estas también pueden ser creadas con vistas a ser desplegadas en

diferentes entornos de computación con lo cual se convierte en un patrón de asignación. **(Bass, Clements, & Kazman, 2013, p. 235)**

- **Contexto:** En un entorno distribuido, existe a menudo la necesidad de dividir la infraestructura del software en diferentes subconjuntos. Las razones pueden ser de operacionales o reglas de negocio**. (Bass, Clements, & Kazman, 2013, p. 235)**
- **Problema:** Cómo dividir el sistema en estructuras de ejecución independientes-grupos de softwares y hardware-conectados por algún medio de comunicación. Esto con la finalidad de proveer un entorno de servidores óptimo para los requerimientos de operaciones y uso de recursos. **(Bass, Clements, & Kazman, 2013, p. 236)**
- **Solución:** Las estructuras de ejecución de muchos sistemas están organizadas como un conjunto de agrupamiento lógico de componentes, estos grupos se les denomina capa. Cada componente dentro de cada capa es organizado de acuerdo con un criterio, este puede ser: el tipo de componente, mismo entorno de ejecución o un mismo propósito de ejecución.

Las capas introducen unas restricciones topológicas que restringe cual componente puede comunicarse con otro componente. Es decir, un componente puede comunicarse con otro dentro que reside dentro de las capas adyacentes. Adicionalmente introduce la restricción del tipo de comunicación entre componentes de capas adyacentes, es decir, algunas requieren comunicación llamada-retorno en tanto que otras están basada en notificaciones de eventos.

2.2 Antecedentes

En las primeras versiones del lenguaje de programación RPG IV se construían programas monolíticos, los cuales estaban compuestos de muchas líneas de código todas estas en un solo archivo fuente, esto generaba grandes unidades de compilación pues el lenguaje no tenía la capacidad de poder manejar las cosas de otra manera. Dicha arquitectura se denomina OPM Original Program Model por sus siglas en inglés, sin embargo, esta es una forma ineficiente de manejar aplicaciones modulares **(Smith, et al., 2000, p. 35)**. Aún para el tiempo en se escribe este trabajo muchos programadores del lenguaje RPG IV siguen usando este modelo sin siquiera notar ese hecho.

A medida que la necesidad de construir aplicaciones escalables surgía, RPG IV construyó un nuevo modelo arquitectónico para la creación y ejecución de programas. A este nuevo modelo que fue incorporado con la versión del sistema operativo OS/400 V2R3 fue denominado ILE (Integrated Language Environment) por sus siglas en inglés. Este nuevo modelo permite al desarrollador escribir y compilar en pequeñas piezas de código y luego enlazar todas ellas en objetos programas que son muy eficientes cuando se llaman entre ellas.

También en tiempo de ejecución permite el uso de una funcionalidad muy importante que se denomina grupo de activación, este permite manejar de forma eficiente los recursos del sistema operativo usados por el trabajo del usuario, mejor manejo de excepciones, mejora el performance de las aplicaciones y la legibilidad de este nuevo estilo modular de construir aplicaciones **(Smith, et al., 2000, p. 35)**. Esto permite aprovechar las nuevas técnicas de ingeniera que se han construido para el desarrollo de software. Sin embargo, estas mejoras no han sido utilizadas en las

empresas que se dedican al desarrollo de software o empresas cuyos negocios hacen uso de ellas.

El mayor problema al momento de aplicar las mejoras que ofrece ILE en la construcción de las nuevas aplicaciones ha sido la falta de comprensión de los conceptos que este tiene, o en su defecto el desconocimiento total de este nuevo modelo de construcción y ejecución de programas por parte de los programadores **(Smith, et al., 2000, p. 21)**. Es así como IBM en el año 2000 publicó un libro dedicado completamente a ILE denominado **Who Knew You Could Do That With RPG IV? A Sorcerer's Guide To System Access and More** en el cual se abordaban todos sus aspectos en profundidad explicando cada detalle y dando ejemplos concretos de la forma de llevarlo a la práctica en la construcción de aplicaciones. Debido que este lenguaje de programación pertenece a lo que se denomina Software Privativo no se desarrolló a la par de los lenguajes de software libre, pues estos aprovecharon todo el ecosistema construido a su alrededor y construyeron técnicas para el desarrollo de software tales como la Arquitectura de Software.

Tal como se ha dicho anteriormente, IBM no planteaba la aplicación de ninguna técnica de desarrollo de software tales como los famosos patrones de diseño usados en el desarrollo de software o arquitectura para aplicaciones tal como lo hacen los lenguajes actuales como Java, Python, PHP, etc., entonces entre los años 2008 y 2010 Scott Klement publicó un documento en el cual explica la manera de usar las nuevas técnicas de desarrollo de software que este campo ofrece usando el lenguaje RPG IV aplicando ILE. En este documento Klement plantea dos patrones arquitectónicos, a saber: el patrón arquitectónico Modelo Vista Controlador MVC y el patrón arquitectónico Orientado a Servicio SOA **(Klement, pág. 13)**. Este es el primer

trabajo que aborda el problema y propone y aplica un práctico de la ingeniería de software usando RPG IV.

IBM se da cuenta que tiene que tomar las cosas más en serio y estar a la par con la vanguardia que va surgiendo con las nuevas herramientas de la ingeniería del Software y se puede decir que da un gran salto para abordar todo ello y publica en el año 2014 otro libro denominado **Modernizing IBM i Applications from the Database Up to The User Interface and Everything in Between**, aquí ya plantea todas las técnicas que la ingeniería del software ofrece para afrontar el vertiginoso y cambiante mundo de los negocios de hoy, por ejemplo la web, la computación Mobile, el cloud-computing seguido por el impacto de las interfaces gráficas de usuario **(Amra, y otros, 2014, págs. 3-5)**. Más adelante en diciembre del año 2016 hace una completa actualización al libro **Who Knew You Could Do That With RPG IV? A Sorcerer's Guide To System Access and More** y lo renombra como **Who Knew You Could Do That With RPG IV? ModernRPG for the Modern Programmer**, en esta nueva edición plantea las cosas de forma totalmente nueva dándole un enfoque moderno del lenguaje de programación llenando las expectativas que muchos programadores esperaban para afrontar el desafío que el mundo cambiante de la industria experimenta día a día **(Diedrich, y otros, 2016, págs. 3-7)**.

2.3 Glosario de Terminos

2.3.1 Análisis de la situación de la etapa de desarrollo de software

Para llevar a cabo un desarrollo de software es necesario comprender la tecnología que se va a usar, y en este punto vamos a abordar la situación desde la perspectiva del uso del lenguaje de programación, específicamente RPG IV, ya que representa un elemento arquitectónico en el diseño general de aplicaciones

para este caso concreto **(Fowler, 2002, pág. 3)**. Dentro del contexto del lenguaje RPG IV para llevar a cabo el desarrollo de una aplicación, este atraviesa las etapas del ciclo de vida del desarrollo del software, sea este proceso un modelo tradicional o ágil; independientemente del proceso elegido, la etapa de diseño es fundamental para el éxito su construcción y calidad. Es la etapa del diseño dónde introduce la calidad en la ingeniería del software **(Pressman, 2010, pág. 212)**.

Por lo tanto, la etapa del diseño de software es fundamental para todo el ciclo del desarrollo del software, de esta surge lo que se denomina la arquitectura del software como uno de los resultados finales de su desarrollo. Es por lo tanto perentorio comprender los conceptos básicos de diseño ya que aplicados correctamente derivan una representación arquitectónica del sistema a desarrollarse y esta a su vez proporciona un marco de trabajo el cual guía al desarrollo del software **(Pressman, 2010, pág. 217).**

Habiendo sentado las bases para el desarrollo de una arquitectura de software que sirva como guía a los programadores, se va a exponer las características del diseño arquitectónico que tienen las aplicaciones actuales construidas usando el lenguaje de programación RPG IV. Una de las causas que no se ha desarrollado un buen diseño arquitectónico es la falta de comprensión de los conceptos del lenguaje y sus características para hacer un desarrollo modular, especialmente el entorno de desarrollo integrado ILE tal como lo menciona Scott Klement **(Klement, pág. 2)**. La falta de interés de muchos desarrolladores por aprender mejores y modernos métodos de ingeniería de software y aplicarlos a su trabajo, esto es muy notorio aun en estos tiempos, la causa fundamental es que muchas

de las personas que se dedican a crear nuevas aplicaciones y dar mantenimiento son programadores de edad muy madura **(Smith, et al., 2000, p. 4)**; estos han trabajado desde los primeros tiempos que el lenguaje RPG IV apareció en el mercado y no han tenido contacto con otras tecnologías que usan modernas técnicas que la industria proporciona.

Como consecuencia de todos los factores antes mencionados el resultado son aplicaciones cuyo diseño arquitectónico; en su mayoría monolítico, no soporta las necesidades de negocios de las empresas, lo cual lleva a que las aplicaciones construidas expongan las siguientes características que su mal diseño arquitectónico posee **(Amra, y otros, 2014, pág. 31)** :

- Rigidez: Hacer cambios a la aplicación es muy difícil ya que un solo cambio podría desencadenar múltiples cambios en cascada. Con un diseño rígido lo que parece ser un cambio sencillo puede convertirse en un proyecto difícil de estimar.
- Fragilidad: Los cambios pueden dañar al sistema en muchas partes cuando uno solo es hecho. Estas partes no tienen relación conceptual con el área que fue cambiada.
- Inmovilidad: Cuando la aplicación contiene partes reusables, sin embargo, el esfuerzo de separarlas del sistema original es demasiado grande. La aplicación está muy enmarañada para reutilizar partes de ella.
- Viscosidad: Cuando se realiza un cambio a una aplicación se tiene dos opciones: preservar el diseño o hacer parches rompiendo el diseño, esto sucede por qué no se entiende el diseño o por presiones de tiempo.

- Complejidad Innecesaria: El diseño muchas contiene elementos que no se utilizan y resultan innecesarios. Esto surge como la anticipación a posibles cambios futuros, sin embargo, se tiene que planificar bien para agregar complejidad al diseño.
- Repetición Innecesaria: El diseño contiene estructuras repetidas que pueden ser unificadas bajo un solo componente. A lo largo del tiempo y por la falta de conocimiento del diseño y la arquitectura los programadores han abusado del "copia-y-pega" que ha llevado a la redundancia de código fuente.
- Opacidad: El diseño y el código fuente son difíciles de leer ya que no expresa su propósito correctamente.

2.3.2 Desarrollo de ILE para el lenguaje de programación RPG IV

2.3.2.1 Entorno de desarrollo integrado ILE (por sus siglas en inglés)

ILE es un entorno en el cual código de muchos lenguajes de programación pueden ser compilados, enlazados, y ejecutados **(Klement, RPG Programming, 2010, pág. 2)**. El sistema operativo "System i" maneja un entorno en el cual se manejan muchas tecnologías como COBOL, C, CL, Java, PHP, Python, C++, entre otras. Estas tecnologías pueden manejarse a la misma vez dentro de este entorno, esto permite que un programa escrito en RPG IV pueda llamar a un método de un programa escrito en Java o en C.

La primera versión de ILE fue introducida en la versión **V2R3** del sistema operativo System i, el cual en ese entonces era llamado **OS/400** en el año 1993 para el lenguaje **C**, luego fue incorporado para el lenguaje

RPG IV en la versión **V3R1** del sistema operativo **OS/400**. Antes de ello existía el modelo de construcción de aplicaciones llamado **OPM (Original Program Model)** por sus siglas en inglés **(Smith, et al., 2000, p. 61)**, este modelo como ya se mencionó anteriormente, generaba grandes unidades de compilación, es decir, programas monolíticos que no permitían hacer uso de las modernas técnicas de desarrollo de software, especialmente construir programaras modulares.

ILE proporciona nuevos tipos de objetos con los cuales se pueden construir aplicaciones más escalables y un mejor manejo de los recursos del sistema operativo con la creación de una estructura dentro del contexto de ejecución de un programa llamada grupo de activación **(Activation Group)**, este grupo de activación es creado cuando un programa ILE es iniciado o activado.

Sin embargo, RPG IV permite hacer uso de ambas técnicas; es decir, como un lenguaje ILE y como uno No-ILE, esto se logra cuando el valor del parámetro **DFTACTGRP(*YES)** (Default Activation Group) tiene valor "*YES" o usado con el valor por defecto con el comando **CRTBNDRPG**, entonces el programa en un modo que es compatible con programas **NO-ILE (Amra, y otros, 2014, pág. 213)**. Es en este concepto muy pequeño, pero de suma importancia que los programadores RPG no prestan la atención debida y terminan por desperdiciar la potencia del lenguaje, pues ILE ofrece características muy importantes que aprovechadas de la forma correcta permite construir excelentes aplicaciones de negocio que requieren procesamiento de transacciones a

gran escala, entre ellas podemos mencionar **(Amra, y otros, 2014, pág. 214)**:

- Modularidad para la construcción de aplicaciones.
- Capacidad de escribir pequeñas y reusables piezas de código.
- Uso de funciones ILE ya creadas que permiten escribir menos código.
- Capacidad de agrupar funciones que son llamadas por múltiples programas en programas de servicio; objetos tipo *SRVPGM.
- Mejor manejo de recursos del sistema operativo.
- Mejor manejo de errores durante la ejecución de un programa

2.3.2.2 Módulos

Los módulos son objetos de tipo *MODULE que son creados por el compilador cuando el comando CRTRPGMOD es usado. Un módulo puede estar compuesto de un procedimiento principal (main) y de uno más sub-procedimientos. Aquellos módulos que no poseen un procedimiento principal son aquellos que están destinados a estar en un programa de servicio **(Diedrich, y otros, 2016, pág. 78)**.

Un módulo es llamado algunas veces una unidad de compilación por qué es producto de la compilación de un miembro fuente; es decir del archivo que contiene el código fuente. Los módulos no son ejecutables por sí mismos, ellos sirven como bloques de construcción para crear programas normales o programas de servicio.

2.3.2.3 Programas de servicio

Los programas de servicio es una de las mejores características que ofrece el entorno ILE, esto se expone en el siguiente escenario en el cual entran a tallar ofreciendo un diseño elegante dada la naturaleza estructurada del lenguaje.

Supongamos que tenemos el escenario en el cual se tiene procedimientos que son llamados por múltiples programas, entonces se pueden enlazar los módulos que contienen dichos procedimientos copiando cada código compilado en cada uno de los programas desde el cual son llamados. Sin embargo, esto tiene muchas desventajas. Por ejemplo, los procedimientos ocuparían espacio en cada programa, y cada programa requeriría ser actualizado cuando el procedimiento sufra cambios, causando problemas de mantenimiento.

En lugar tomar el enfoque anterior, se pueden agrupar dichos procedimientos en un programa de servicio. Entonces los procedimientos están implementados en un solo lugar y son compartidos por los programas desde los cuales son llamados, lo que mejora considerablemente el mantenimiento de las aplicaciones. Por lo tanto, podemos definir a un programa de servicio como una colección de módulos, especialmente aquellos que contienen procedimientos. Son objetos de tipo ***SRVPGM**, estos no pueden ser llamados directamente, sin embargo, los procedimientos que este contiene pueden ser llamados por programas **ILE (Diedrich, y otros, 2016, pág. 78)**.

2.3.2.4 Enlazamiento (Binding): creación de programas *PGM y programas de servicio *SRVPGM

Los programas ILE y los programas de servicio son creados utilizando un procedimiento denominado enlazamiento(***binding***). El procedimiento de enlace copia el código compilado de objectos modulo (objetos ***MODULE**), ya sea para crear programas enlazados; esto es objetos tipo ***PGM**, o programas de servicio; objetos tipo ***SRVPGM (Diedrich, y otros, 2016, pág. 79)**.

Un programa ILE puede ser creado de uno o más objetos módulos (***MODULE**) usando el comando Create Program (**CRTPGM**), por sus siglas en inglés. Además, el comando **CRTPGM** puede enlazar al programa a ser creado uno más programa de servicio. Los módulos cuyo código va a ser copiado en el programa a crear (usualmente este programa sólo tiene un módulo entrada) son listados en el parámetro ***Module*** del comando **CRTPGM**. Los programas de servicio que va a ser referenciados por este programa y usados en tiempo de ejecución (estos no son copiados dentro del objeto programa) son listados en el parámetro ***Service Program*** del comando **CRTPGM**, sin embargo, el último proceso descrito puede ser simplificado a través del uso de directorios de enlace **(Diedrich, y otros, 2016, pág. 79)**.

Los programas de servicio son creados usando el comando Create Service Program (**CRTSRVPGM**) por sus siglas en inglés, de la misma forma que para crear programas tipo ***PGM**, los programas de servicio contienen código que es copiado de distintos objetos módulos (*MODULE), estos referenciados en el parámetro ***Module*** del comando,

y también pueden referenciar a otros programas de servicio a, estos referenciados en el parámetro ***Service Program*** de dicho comando **(Diedrich, y otros, 2016, pág. 79)**.

Los programas RPG IV puede ser creados ejecutando el comando Create Bound RPG Program (**CRTBNDRPG**). Este comando puede ser ejecutado sobre el miembro de código fuente que contiene el módulo de entrada para el programa, este es el comando usado usualmente por los programadores ignorando que sucede en su ejecución. Este comando ejecuta dos acciones en un solo paso: la compilación y el enlazamiento **(Diedrich, y otros, 2016, pág. 79)**. Si el programa a ser creado necesita de un módulo o de un programa de servicio para ser creado entonces necesita de un directorio de enlace, concepto que va a ser explicado más adelante.

2.3.2.5 Directorios de enlace

Los directorios de enlace son objetos de tipo *BNDDIR, un directorio de enlace contiene una lista de módulos y programas de servicios que son candidatos para ser enlazados en un programa que los necesita. Los directorios de enlace pueden ser especificados en los comandos **CRTPGM** o en el comando **CRTBNDRPG**. Sin embargo, la manera más fácil de usarlos con código RPG IV es referenciarlos en la especificación de control del miembro de código fuente **(Diedrich, y otros, 2016, pág. 79).**

No todos los ítems que están contenidos en el directorio de enlace son necesariamente enlazados. Sólo aquellos ítems que contienen procedimientos que son llamados (aquellos que son importados) son

enlazados. Los módulos y programas de servicio que están listados en un directorio de enlace a menudo contienen procedimientos estándar, tales como funciones matemáticas y otras del sistema. Los comandos para su mantenimiento son los que se listan a continuación.

- Crear directorio de enlace **CRTBNDDIR** (Create Binding Directory), por sus siglas en inglés.
- Agregar entrada al directorio de enlace **ADDBNDDIRE** (Add Bindign Directory) por sus siglas en inglés.
- Trabajar con directorio de enlace **WRKBNDDIR** (Work with Binding Directory).
- Trabajar con entrada de directorio de enlace **WRKBNDDIRE** (Work with Binding Directory Entry) por sus siglas en inglés.

2.3.2.6 Exportar programas de servicio

Los programas de servicio poseen un mecanismo de encapsulamiento que les permite proteger el acceso a los procedimientos de los módulos de los cuales está compuesto, similar al concepto del paradigma de la programación orientada a objetos, este mecanismo que protege el acceso a sus procedimientos es denominado ***export*** por sus siglas en inglés **(Diedrich, y otros, 2016, pág. 80)**, esto significa que hace visible un subprocedimiento para ser invocado por programas que lo necesitan y a los cuales han sido enlazados el programa de servicio al cual pertenece.

Para hacer exportable un procedimiento, este tiene que haber declarado en su definición la palabra clave ***EXPORT***. Si un módulo llega a formar parte de un programa de servicio sus procedimientos pueden ser exportados desde el programa de servicio para estar disponibles a

programas externos o a otros programas de servicio. Esto quiere decir, que el procedimiento tiene que ser exportado desde el módulo; usando la palabra clave ***EXPORT***, desde el cual forma parte y desde el programa de servicio que llega a formar parte dicho módulo.

Existen dos maneras de exportar procedimientos desde un programa de servicio, ambas controladas por el uso de la palabra clave ***EXPORT*** sobre el comando **CRTSRVPGM**.

Si el valor es *ALL para la palabra clave ***EXPORT***, entonces todos los procedimientos que han sido exportados desde los módulos del programa de servicio son exportados automáticamente. Los procedimientos llegan estar disponibles para otros programas u otros programas de servicio.

Si el valor de ***EXPORT*** tiene especificado ****SRCFILE***, entonces un miembro de código fuente que contiene un lenguaje de enlazado es necesario, este lenguaje de enlazado declara cuales procedimientos deben a ser exportados. Además de módulos y procedimientos, se pueden exportar variables, sin embargo, dicha práctica no es recomendable ya que es propensa de errores.

2.3.2.7 Lenguaje de enlazado

El comando crear programa de servicio **CRTSRVPGM** (Create Program Service) por sus siglas en inglés, especifica cómo el programa de servicio es enlazado y que procedimientos y variables exporta. El parámetro ***EXPORT*** del comando específica la manera cómo los procedimientos exportados son disponibles a usuarios externos.

El parámetro **EXPORT** cuyo valor por defecto es *SRCFILE requiere un miembro especial de código fuente que exista. El miembro de código

fuente, denominado miembro de código fuente de lenguaje de enlazado, contiene la lista de los nombres de los procedimientos que va a ser exportados, y posiblemente variables, que el programa de servicio los hace disponibles. Los otros procedimientos que tienen en su declaración la palabra clave EXPORT permanecen inaccesibles para el resto de los usuarios externos **(Diedrich, y otros, 2016, pág. 80)**. La miembro fuente es de tipo **BND** y está localizado en un archivo fuente, cuyo valor por defecto es **QSRVSRC**, esta miembro fuente nunca es compilado.

Un valor ****ALL*** para el parámetro **EXPORT**, esto es ***EXPORT(*ALL)***, hace disponible todo todos procedimientos que tienen la palabra **EXPORT** en el programa de servicio. Este enfoque presente una gran desventaja, pues la firma del programa del servicio es actualizada cada vez que se agrega un nuevo procedimiento a este, con lo cual los programas que han enlazado dicho programa de servicio muestran errores en tiempo de ejecución. Por lo tanto, es necesario actualizar cada programa que enlazó el programa de servicio para que también actualice la nueva firma, este enfoque resulta engorroso ya que la labor para hacerlo si tiene muchos programas es demasiado costosa **(Diedrich, y otros, 2016, pág. 80)**.

Por ello el uso del lenguaje enlazado permite conservar la firma del programa de servicio sin que afecte a todos los programas que lo enlazaron no causando errores por qué estos ya no encuentra la firma del programa de servicio con la cual fueron creados.

2.3.2.8 Grupos de activación

Una de las características fundamentales que proporciona ILE, es que permite administrar los recursos del sistema operativo a través de un mecanismo denominado grupos de activación, es decir que cuando se carga un programa, estos se les pueden asignar un espacio de memoria de trabajo dentro del trabajo en el cual se están ejecutando, aislando el contexto de su ejecución, evitando colisión con otros programas que se ejecutan dentro del mismo trabajo en el sistema operativo ***System i***. Estos grupos de activación son estructurales de almacenamiento temporales. Existen tres tipos de ellos: grupo de activación por defecto (***default***), con nombre, (***named***), y nuevo (***new***).

Los grupos de activación por defecto (***default activation groups***) existen automáticamente y nunca se borran, existen dos clases de este tipo de grupos activación: grupo de activación por defecto 1 y grupo de activación por defecto 2, el primero es usado por muchas funciones del sistema operativo, y el segundo por programas ILE que son creados con el valor ****YES*** en el parámetro ***DFTACTGRP(*YES)*** del comando **CRTBNDRPG**.

Los otros dos tipos de grupo de activación son especificados por el valor del parámetro ***ACTGRP*** en los comandos de creación de un programa y de un programa de servicio **CRTPGM** y **CRTSRVPGM** respectivamente. Por lo tanto, el grupo de activación es determinado por el programa o programa de servicio al momento de su creación.

Un grupo de activación es creado cuando el programa es iniciado, este puede incluir lo siguiente:

- Variables estáticas y automáticas

  Las variables de programas ejecutándose en grupos de activación. Las variables estáticas son aquellas definidas en el procedimiento principal, estas provienen de fuentes externas tales como, DDS y especificaciones SQL o estas pueden ser definidas como variables RPG. Las variables automáticas son aquellas definidas en los sub-procedimientos.

- Rutas hacia archivos abiertos (**ODP Open Data Path**), por sus siglas en inglés.

  Estos son objetos temporales que representan rutas temporales de archivos hacia programas que necesitan de la información que estos necesitan para realizar su trabajo. Esto son buffer de información y punteros hacia registros como parte de las rutas abiertas hacia esos archivos.

- Almacenamiento asignado dinámicamente

  Objetos creados temporalmente por la operación **ALLOC** en el programa RPG IV.

- Rutina de manejo de errores

  Manejo de mensajes de error de los programas de usuario o del sistema a través, de la implementación de módulos personalizados para manejarlos. Estos provienen de la pilada de llamadas independientemente del lenguaje de programación en que se hallan escrito.

2.3.2.9 Comandos CL usados con ILE y RPG IV

Hay cuatro comandos básicos para la creación de módulos **ILE**, programas y programas de servicio, a continuación, se exponen cada uno de ellos:

- Comando Crear módulo **RPG, CRTRPGMOD** (Create RPG Modulo) por sus siglas en inglés.

  Este comando invoca al compilador ILE RPG, el cual produce el objeto tipo ***MODULE**.

- Comando Crear Programa, **CRTPGM** (Create Program) por sus siglas en inglés.

  Este comando invoca al enlazador **ILE**, independientemente del lenguaje de programación, y crea el objeto ***PGM** de los módulos especificados y de los programas de servicio.

- Comando Crear Programa de Servicio, **CRTSRVPGM** (Create Service Program) por sus siglas en inglés.

  Invoca al enlazador ILE, independientemente del lenguaje de programación, y crea el objeto de tipo *SRVPGM de los módulos especificados y otros programas de servicio.

- Comando Crear Programa RPG Enlazado, **CRTBNDRPG** (Create Bound RPG Program) por sus siglas en inglés.

  Este comando crea un módulo en la librería **QTEMP** y crea un objeto programa de tipo ***PGM** de dicho módulo. Este módulo se pierde luego que el trabajo termina.

  Otros comandos que permiten cambiar, visualizar, y borrar funciones:

- Comando Mostrar Módulo, **DSPMOD** (Display módulo) por sus siglas en inglés.

  Muestra información al usuario del módulo en la pantalla o en la salida de impresora.

- Comando Mostrar Programa de Servicio, **DSPSRVPGM** (Display Service Program) por sus siglas en inglés.

  Muestra información acerca del programa de servicio en la pantalla del usuario o en la salida de impresora.

- Comando Mostrar Programa, **DSPPGM** (Display Program) por sus siglas en inglés.

  Muestra información del programa en la pantalla del usuario o en la salida de impresora.

  Los siguientes comandos permiten trabajar con directorios de enlace para luego ser referenciados por el comando **CRTSRVPGM** como fuente de procedimientos aptos para ser exportados.

- Comando Crear Directorio de Enlace, **CRTBNDDIR** (Create Binding Directory).

  Crea un objeto de tipo ***BNDDIR** para luego ser especificado o referenciado por el parámetro ***BNDDIR*** de el comando **CRTSRVPGM**.

- Comando Agregar Entrada al Directorio de Enlace, **ADDBNDDIRE** (Add Binding Directory Entry) por sus siglas en inglés.

Este comando agrega entradas al directorio de enlace, sean estos objetos de tipo ***MODULO**, es decir módulo o programas de servicio, es decir objetos de tipo ***SRVPGM**.

- Comando Trabajar con Entradas de Directorio de Enlace, **WRKBNDDIRE** por sus siglas en inglés.

  Este comando muestra información de las entradas que contiene un directorio de enlace, sean estos módulos o programas de servicio, de forma tal que se le puedan dar mantenimiento, ya sea agregar, eliminar una entrada de un directorio de enlace.

2.4 Hipótesis

2.4.1 Hipótesis general

El diseño e implementación de una arquitectura software para el lenguaje de programación estructurado RPG mejorará el proceso de desarrollo de aplicaciones empresariales.

2.4.2 Hipótesis específica

H1: Es posible mejorar el proceso de desarrollo de desarrollo de aplicaciones empresariales usando el lenguaje de programación estructurado RPG.

H2: Es posible aplicar un patrón arquitectónico que sirva como marco de trabajo de para el desarrollo de aplicaciones empresariales usando el lenguaje de programación estructurado RPG.

Capítulo 3: Marco metodológico

3.1. Tipo y nivel de investigación

El presente trabajo se considera una Investigación Aplicada Fundamental, ya que su propósito primordial es utilizar las mejores técnicas de la ingeniería de software, específicamente de la arquitectura de software para mejorar la construcción de aplicaciones empresariales usando una lenguaje de programación estructurada.

3.2. Modelo teórico

Para poder realizar la implementación de esta arquitectura de Software se va a usar el ciclo de desarrollo de Software Iterativo, el cual consiste en mejorar en cada iteración el desarrollo de un sistema con la entrega de componentes más avanzados en cada una de sus interaciones. Cada iteración consta de las etapas análisis, diseño, implementación y pruebas. Durante la etapa de implementación y diseño de la arquitectura se usará el Lenguaje Unificado de Modelado UML 2.0 que permitirá documentar cada componente de la arquitectura de software a implementar.

Se empezará por analizar y documentar la arquitectura de una aplicación existente que haya sido escrita con el lenguaje de programación RPG, cuál son los objetivos de negocio que cumplen para la empresa y los problemas que esta tiene poder ampliar los requerimientos de negocio que los usuarios solicitan. Luego de ellos se empezará a diseñar la arquitectura diseñando los componentes basándose en las herramientas que proporciona el lenguaje de programación.

La arquitectura que se desarrolle será desplegada sobre un servidor IBM-i que es desarrollado por la empresa IBM, esta tecnología es usada ampliamente por grandes empresas que se dedican a manejar grande volúmenes de información, especialmente aquellas que se dedican al negocio finaciero.

3.3. Diseño de investigación

El diseño de la investigación es experimental, ya que identifica las características que las controla, las altera o manipula con el fin de observar los resultados.

Una investigación con diseño experimental es una estructura dónde al menos se manipula una variable y las unidades son asignadas aleatoriamente a los distintos niveles o categorías de la variable o variables manipuladas.

3.4. Diseño de técnicas e instrumentos de recolección de datos e información

- **Encuestas**: La encuesta es una técnica que consiste en obtener información acerca de una parte de la población o muestra, mediante el uso del cuestionario o de la entrevista.
- **Cuestionario:** Es un formato redactado en forma de interrogatorio para obtener información acerca de las variables que se investigan, puede ser aplicado personalmente o por correo y en forma de individual o colectiva y debe reflejar y estar relacionado con las variables y sus indicadores.
- **Análisis Documental:** Revisión de todos los documentos con que se cuentan para poder realizar un análisis de la situación que se va a desarrollar.

3.5. Tipo de técnicas de muestreo

- **Población**

  Las aplicaciones Software que existen en el mercado desarrolladas con el lenguaje RPG.

- **Unidad de Análisis**

  La áreas usuarias de las aplicaciones, el equipo programadores que se encargan de su creación y mantenimiento, otros desarrolladores que nunca han usado el RPG como lenguaje de desarrollo.

3.6. Métodos, técnicas y uso de software de tratamiento de análisis de datos

La presente investigación va a realizar algunos cálculos estadísticos, por ello será necesario el uso de algún software que sirva herramienta para llevar a cabo dicha tarea, para ello se va a usar IBM SPSS Statistics. Este software está provisto de una amplia variedad de capacidades analíticas. Acceso descriptivo estadístico, regresión lineal y presentación de gráficos.

Capítulo 4: Aplicación de la propuesta de la arquitectura de software

4.1 Requerimiento de usuario para el módulo de créditos.

Área solicitante: Operaciones/Crédito.

Responsable: La jefa del área de Operaciones.

Usuario de Pruebas: Asistente de Operaciones Empresa Financiera.

1. Objetivo General: Automatizar Check-List de Operaciones de acuerdo con la normativa vigente.
2. Beneficios y justificación:
   - Desembolso más rápido de los créditos.
   - Control de registro de la verificación del expediente físico al momento del desembolso.
3. Descripción del requerimiento.

3.1 El requerimiento se compone de una lista de condiciones a través de preguntas **(ver Anexo A)**, hechas al usuario por parte del asistente de operaciones. Estas preguntas aparecen al momento de realizar el desembolso del crédito: antes y durante esta etapa. Cuando el asistente de operaciones responde a las preguntas del Check-List antes del desembolso, el sistema valida que todas las preguntas estén respondidas mostrando luego una ventana al usuario con la pregunta: **¿Documentación verificada es conforme y se encuentra en el expediente?**, si el asistente de operaciones responde **SÍ**, el sistema permite realizar el desembolso. Esto permitirá minimizar el riesgo de fraude y asegurar que la documentación esté correcta, las condiciones a tratar son las siguientes:

- Se modificará la transacción de desembolso de crédito, transacción **6525**, este cambio consiste en mostrar una serie de preguntas que el asistente de operaciones deberá hacer al cliente antes de realizar el desembolso de crédito y durante dicha operación, las preguntas se van a mostrar según el desembolso sea en presencia de cliente o no.
- La documentación es conforme si las preguntas 1, 3,4,5 y 6 (Ver **Anexo A**) están marcada con SÍ, tanto para persona jurídica como persona natural. La pregunta número 2 sólo estar respondida con SÍ para persona jurídica.
- En caso las respuestas no estén alineadas, es decir algunas están respondidas con SÍ y otras con NO y la respuesta la pregunta **¿Documentación verificada es conforme y se encuentra en el expediente?** es sí, el sistema debe mostrar un mensaje de error: **"Respuestas del Check-List no concuerdan, Revisar"**, presionar la tecla **ENTER** para aceptar el mensaje y el sistema regresará a la pantalla anterior.
- En el caso las respuestas a las preguntas se encuentran alineadas, el sistema no debe permitir responder con NO a la pregunta: **¿Documentación verificada es conforme y se encuentra en el expediente?**, el sistema debe de mostrar el siguiente mensaje: **"Respuestas al Check-List no Concuerdan, Revisar"**, el usuario presiona la tecla ENTER para aceptar el mensaje y el sistema le muestra al asistente de operaciones la pantalla anterior para que corrija las respuestas si es el caso.

- El sistema aceptará como respuesta NO **(NO = 0)** a la pregunta: **¿Documentación verificada es conforme y se encuentra en el expediente?** cuando la pregunta 1 haya sido respondida con NO (**NO = 0)**; si el cliente es persona natural la respuesta a la pregunta 2 también será no, el sistema generará una alerta que va a ir a la bandeja del administrado de agencia Empresa Financiera que sólo él podrá levantar.
- El sistema debe aceptar respuesta NO (**NO = 0**) a la pregunta: **¿Documentación verificada es conforme y se encuentra en el expediente?** cuando se da el caso que el asistente de operaciones ha respondido a las preguntas 3, 4, 5, o 6 (Persona Natural), o 2, 3, 4, 5 o 6 (Persona Jurídica) con NO (**NO = 0**), para esto el sistema debe de mostrar el mensaje **"Respuestas al Check-List no Concuerdan, Revisar"**, el asistente de operaciones deberá presionar la tecla **ENTER** y el sistema le mostrará la pantalla anterior para que corrija las respuestas.

3.2 Si se grabó con NO (**NO=0**) el Check-List, después que el administrador levante las restricciones el **Check-List** antes del desembolso debe de grabarse con repuesta **SÍ**, es decir ya debe de estar concluido y no volver a mostrarle al asistente de operaciones para que este lo llene y lo grabe.

3.3 Cuando las respuestas son satisfactorias al **Check-List** antes del desembolso y el asistente de operaciones presiona la tecla **F5** (Grabar respuestas en el sistema), el sistema debe de mostrar el mensaje **¿Desea Grabar lo Verificado?**

3.4 En el Check-List durante el desembolso, cuando las respuestas son satisfactorias (SÍ =1) y el asistente de operaciones presiona la tecla F5 (Grabar respuestas en el sistema) el sistema debe de mostrar el mensaje: **¿Desea Continuar con el Desembolso?**

3.5 Cuando el administrador de agencia ha levantado las restricciones del Check-List previo al desembolso, en la pantalla de mensajes de mensajes de restricciones debe mostrarse el mensaje: **"Check-List Previo propuesta N° propuesta Cliente Código Aprobado"**.

3.6 Incluir en la pantalla de mensajes y restricciones (pantalla R7102310) la opción de desembolso presencial y no presencial; con la finalidad que el **Check-List** durante del desembolso se muestre al asistente de operaciones si este selecciona **SÍ** en el sistema y no se muestra en caso contrario.

3.7 Luego de grabar exitosamente el **"Check-List Antes del desembolso"** el sistema muestra la pantalla **R7102310** cuya última columna de la grilla que muestra tiene por encabezado: **"CL P."** (Check-List de Propuesta) si ésta tiene como valor **"SI"** el asistente de operaciones se sitúa sobre dicha fila y al momento de seleccionarla presionando la tecla **ENTER** sobre esta, el sistema debe mostrar una pantalla con el siguiente mensaje: **"Existen Mensajes que Deben de Ser Atendidos Antes del Desembolso"**, el asistente de operaciones acepta el mensaje presionando la tecla ENTER e inmediatamente el sistema muestra la pantalla de mensajes de restricciones informando lo siguiente: **"Check-List durante el desembolso NO"**, adicionalmente también muestra las otras

restricciones del crédito, excepto aquellas que indican que existen pagares por cancelar.

3.8 En la etapa del **Check-List** durante el desembolso, el sistema **SOFIA** debe realizar ciertas validaciones a las respuestas que registra el asistente de operaciones tal como se hace en el **Check-List** antes del desembolso, es decir, estas tienen que ser consistentes.

- El desembolso es conforme si (respuestas están alineadas): la respuesta a la pregunta 1 es **SI** y la respuesta a las preguntas 2 y 3 es **NO**.
- Si las preguntas no están alineadas, el sistema no debe permitir responder con **SI** a la pregunta **¿Se puede realizar el desembolso?**, mostrando el mensaje **"Respuestas del Check-List no concuerdan, Revisar"** el asisten de operaciones acepta el mensaje presionando la tecla ENTER sobre este e inmediatamente el sistema regresa a la pantalla anterior, con la finalidad que el asistente de operaciones corrija sus respuestas.

3.9 En la etapa del desembolso, pantalla **R7102310**, en la grilla que se ha descrito en el punto 3.7, campo de la grilla **"CL P"** (Check-List Previo), si la respuesta a este campo es **NO**, la fila de esta grilla debe mostrarse en color rojo, y si es satisfactorio cambiar a otro color.

3.10 Crear una nueva opción en el menú colocaciones del administrador que le permita levantar las restricciones de **Check-List** hechas a las propuestas de desembolso.

3.11 Crear una nueva opción en el menú de administrador regional, jefe de créditos Empresa Financiera, administrador de agencia, supervisor de

operaciones que les permitan consultar las clientes cuyas restricciones le hayan sido levantadas.

3.12 En la pantalla de R71C0018 (pantalla de **Check-List** durante el desembolso) se debe de guardar cada Check-List como histórico independiente de la respuesta a la pregunta: **¿Se puede Realizar el Desembolso?**, haya sido SI o No. Si el usuario del sistema SOFIA ingresa por segunda vez debe de crear un nuevo Check-List basado en las respuestas conservando todos los datos del Check-List que se llenó previamente.

3.13 El sistema SOFIA debe validar que si el asistente de operaciones realizó satisfactoriamente el Check-List durante el desembolso, pero este no llega a efectuarse, cuando el mismo asistente de operaciones u otro usuario ingrese a efectuar el desembolso el Check-List durante el desembolso debe de volver a mostrarse.

4.2 Aplicación de la propuesta del marco de trabajo a empresa financiera

En este capítulo se va a exponer la aplicación del marco de trabajo al Sistema de Software Financiero Automatizado, **SOFIA** de una Empresa Financiera, se detalla la situación en la que se hacía el trabajo antes de aplicar dicho marco de trabajo arquitectónico y la forma cómo ayudó a resolver el problema.

La Empresa Financiera divide su sistema **SOFIA** en módulos por productos que ofrece a sus clientes, para este caso particular, se desarrolló para el módulo de créditos bajo la forma de un requerimiento de parte del área de operaciones-créditos al área de sistemas.

El escenario que propone este requerimiento es ideal para mostrar las debilidades o síntomas del mal diseño que las aplicaciones de Empresa

Financiera tenían para entonces y que aún continúan construyéndolas a la antigua usanza, la forma cómo muchos programadores abordaban una solución a los diferentes cambios que los usuarios solicitaban y se expone con detalle la aplicación del nuevo enfoque en la adecuación del módulo de créditos para adicionar una funcionalidad para cumplir una norma de la SBS (SUPERINTENDENCIA DE BANCA, SEGUROS Y AFP).

La figura 4.2.1 muestra el diagrama de arquitectura que resultará al integrar el nuevo marco arquitectónico propuesto con el sistema existente, SOFIA, cuya arquitectura es monolítica.

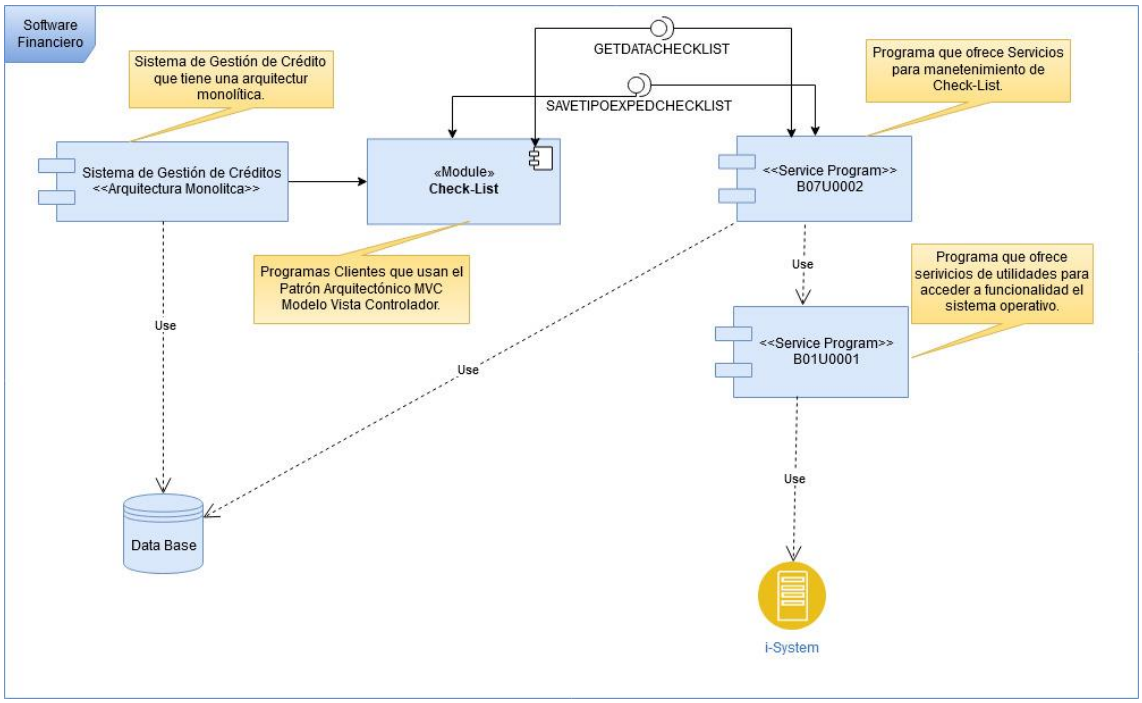


*Figura 4.2.1.*Diagrama de Arquitectura del nuevo Marco Propuesto en interacción con el existente.

4.2.1 Programas del módulo desembolsos de Propuestas de créditos empresa financiera.

El lenguaje de programación RPG cuenta con una característica única y poderosa, la cual consiste en la capacidad acceder a los archivos de base de datos directamente con los cuales se desea trabajar, esto se logra simplemente declarándolos dentro del programa que hará uso de ellos, ya que la base de datos forma parte integral del sistema operativo System i sobre el cual se ejecuta el programa RPG, esta base de datos es **DB2**. Sin embargo, esta ventaja también representa una trampa para muchos programadores que terminan haciendo un uso inapropiado de ella al no aplicar buenas prácticas de ingeniera de software al momento de desarrollar una aplicación.

Debido a la poca claridad con que se explican los conceptos fundamentales del lenguaje de programación RPG IV en la documentación que ofrece la empresa IBM o tal como lo expresa Scott Klements, estos resultan ser complicados de comprender para muchos programadores (Klement, presentations, pág. 3), que el resultado que se tiene como diseño final es el mismo que se encuentran en las aplicaciones que heredan a las cuales se encargan de darles mantenimiento. Es decir, tenemos aplicaciones monolíticas las cuales atiborran en un solo archivo de código fuente todas las funcionalidades del programa mezclada con la lógica de negocio, lo cual hace ilegible entender las cosas que hace cada programa, esto representa una labor titánica para el programador que enfrenta la tarea de modificar sus funcionalidades o agregar nuevas.

Estos problemas demandan de tiempo y dinero para las empresas que soportan sus negocios en esta clase de aplicaciones, pues no puede atender rápidamente a

las necesidades que sus clientes requieren y sobre todo con el riesgo latente de incurrir en grandes errores o pérdidas económicas por la poca flexibilidad que ofrecen estas. Esto se puede ver expresado en cada uno de los programas que se expone a continuación, cuál es su tarea principal, las falencias que poseen y la manera cómo se adaptó el nuevo requerimiento a lo que ya se tenía para no agregar más complejidad a estos programas:

- **Programa R7102310**: Este programa pertenece al módulo del sistema de gestión de créditos de la Empresa Financiera, se encarga de resolver los desembolsos para cada propuesta de crédito que tiene asociada un cliente. El requerimiento del área de créditos solicitaba modificarlo para poder agregar el **Check-List** que ya se ha mencionado en el punto 3, la gran dificultad a la que se ha enfrentado es saber en qué punto de toda la lógica que maneja el programa implementar lo solicitado, la solución obvia que se podía haber tomado es seguir el diseño que ya se tenía, y la otra era abstraer toda la nueva funcionalidad en un nuevo diseño que luego permita escalar la aplicación y aplicar todas las técnicas que el lenguaje de programación RPG ofrece, es decir tomar las facilidades de ILE para hacer un mejor diseño al que se tenía e ir cambiando de a pocos aquellos que ya existían; muchos de estos monolíticos.

  Este programa usa el estilo de codificación RPG encolumnado, por lo que se decidió seguir con tal estilo con la finalidad de no perjudicar el diseño original. Este estilo es muy estricto con las reglas para la declaración de sentencias, cada palabra clave del lenguaje debe de ir en cierta posición de acuerdo con las reglas del compilador. Está compuesto por subrutinas en la que se encuentran plasmadas las reglas de la lógica del negocio, y en este

tipo de diseño no existe lo qué es concepto de variables locales, como se conoce en otros lenguajes de programación, sino que por el contrario cada una de ellas posee un ámbito global lo cual hace su uso muy riesgos y el mantenimiento se hace muy engorroso, complicado y tiene un elevado costo de mantenimiento. La figura 4.2.1.1 muestra las subrutinas que componen el programa del cual estamos tratando, para ir analizando detalle a detalle los errores de diseño que expone:

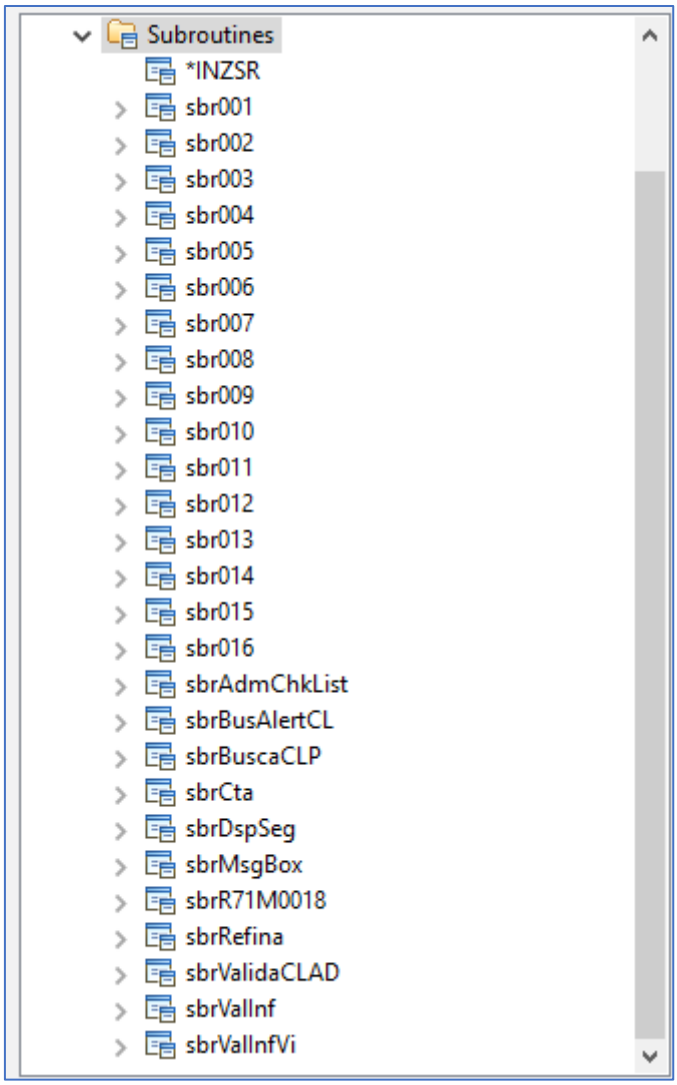


Figura 4.2.1.1. Subrutinas que componen el programa R7102310.

La figura 4.2.1.1 muestra las subrutinas de las cuales está compuesto el programa **R7102310**, como podemos notar, las primeras de ellas no poseen nomenclatura que describa su propósito, lo cual significa la total falta del uso de buenas prácticas. Sin embargo, las últimas expresan en su nombre su funcionamiento, con lo cual se puede intuir cuál es el propósito de cada una. La figura 4.2.1.2 muestra la implementación ***sbrBusAlertCL***.

```
c*-------------------------------------------------------------*
c* verifica si el expediente ha sido observado CLP (Alerta) *
c*-------------------------------------------------------------*
c     sbrBusAlertCL BegSr
c
c                   Eval      nFlgAlert = 0
c                   Move      REPCRE        PROPNRO
c                   Eval      nTipoCL = CHKLIST_PREVIO_DESEMBOLSO
c                   Eval      nEstCL = CHKLIST_ACTIVO
c                   Eval      nFlgSitCL = 2
c
c     cla005        Chain     L805401
c                   If        %Found(L805401)
c     cla001        Setll     F7226
c     cla001        Reade     F7226
c                   Dow       Not %Eof()
c                   If        'CL PREVIO' = %TRIM(%SUBST(MENSAJ:33:9))
c                   If        FLGRES=2 and USULEV='' and MESLEV=0
c                   Eval      nFlgAlert = 1
c                   EndIf
c                   EndIf
c     cla001        Reade     F7226
c                   EndDo
c
c                   EndIf
c
c                   EndSr
c*-------------------------------------------------------------*
```

Figura 4.2.1.2. Implementación de la subrutina sbrBusAlertCL del programa R7102310.

La subrutina ***sbrBusAlertCL*** es una de las que se agregó para el requerimiento solicitado, esta se encarga de buscar aquellas propuestas de desembolsos de

crédito que tienen asociadas un Check-List con algunas restricciones y encender una bandera para indicar que la propuesta las tiene y que no han sido levantadas. El lenguaje de programación RPG en su estilo encolumnado tiene una restricción para los nombres de las subrutinas de 12 caracteres, por lo cual se trató de simplificar para cumplir esta regla y dar claridad a su lectura sin que perdiera claridad en su descripción.

Ahora se va a mostrar un componente importante del programa **R7102310**, este es la pantalla con la cual trabaja para captar las interacciones con el usuario cuyo nombre es **P7102310**; es importante notar que las pantallas que los programas manejan tienen el nombre del programa, pero se cambia la R inicial del nombre por la letra P tal como se puede notar en el nombre que se acaba de señalar. Para indicar las modificaciones realizadas se va a mostrar la vista de diseño y el resultado de aplicar los cambios para satisfacer las necesidades del usuario. Las figuras 4.2.1.3 y 4.2.1.4 muestran lo que se acaba de mencionar, el diseño y el resultado respectivamente.

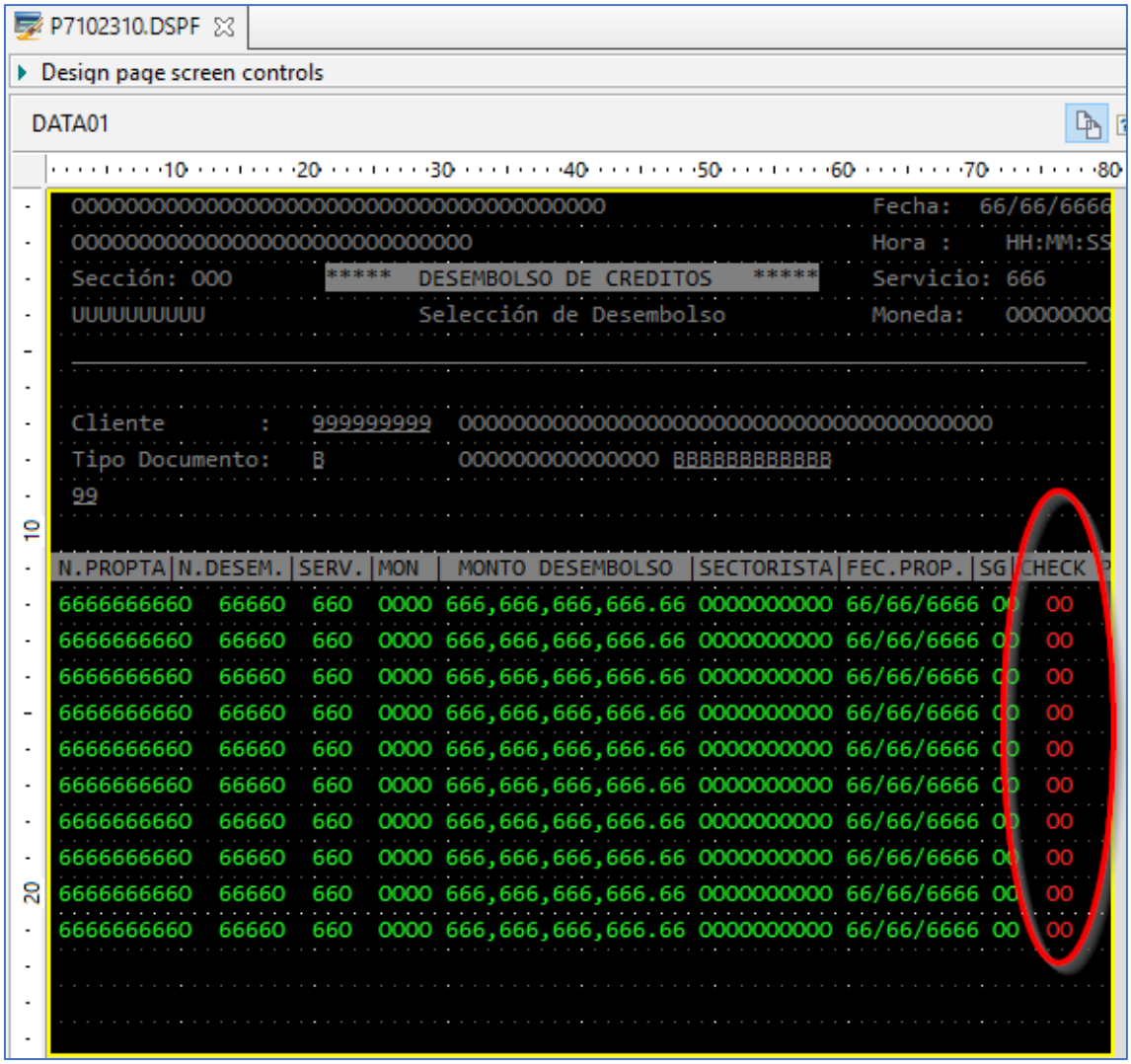


Figura 4.2.1.3. Vista De Diseño Pantalla P7102310 Del Programa R7102310.

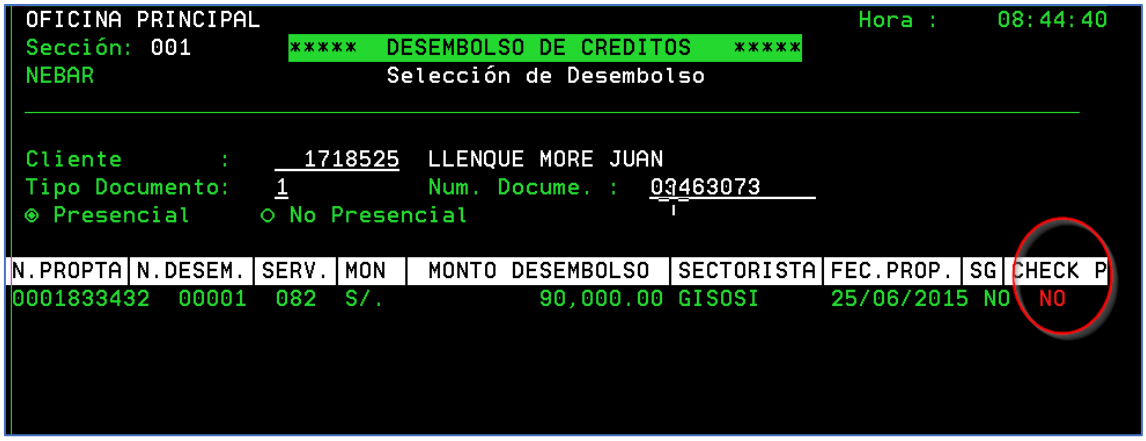


Figura 4.2.1.4. Vista De Ejecución De Pantalla P7102310 Del Programa R7102310.

Las figuras 4.2.1.3 y 4.2.1.4 muestran la modificación realizada a la pantalla **P7102310** para adecuarla y mostrar una columna adicional a su grilla, en este caso **"CHECK P"** que sirve para indicar si la propuesta de desembolso tiene asociado un Check-List previo o no, para resaltar esto se muestra en color rojo indicando la importancia de realizar esta acción.

Ahora vamos a exponer la complejidad a la que se ha enfrentado para poder hacer las adecuaciones que el requerimiento planteaba, esto lo vamos a exponer a continuación en la figura 4.2.1.5, que muestra un diagrama completo del flujo de las llamadas internas que hace a cada subrutina de la cual está compuesto y a programas externos, principalmente al programa **R71M0018** que es dónde se ha llevado toda la lógica para la implementación del Check-List.

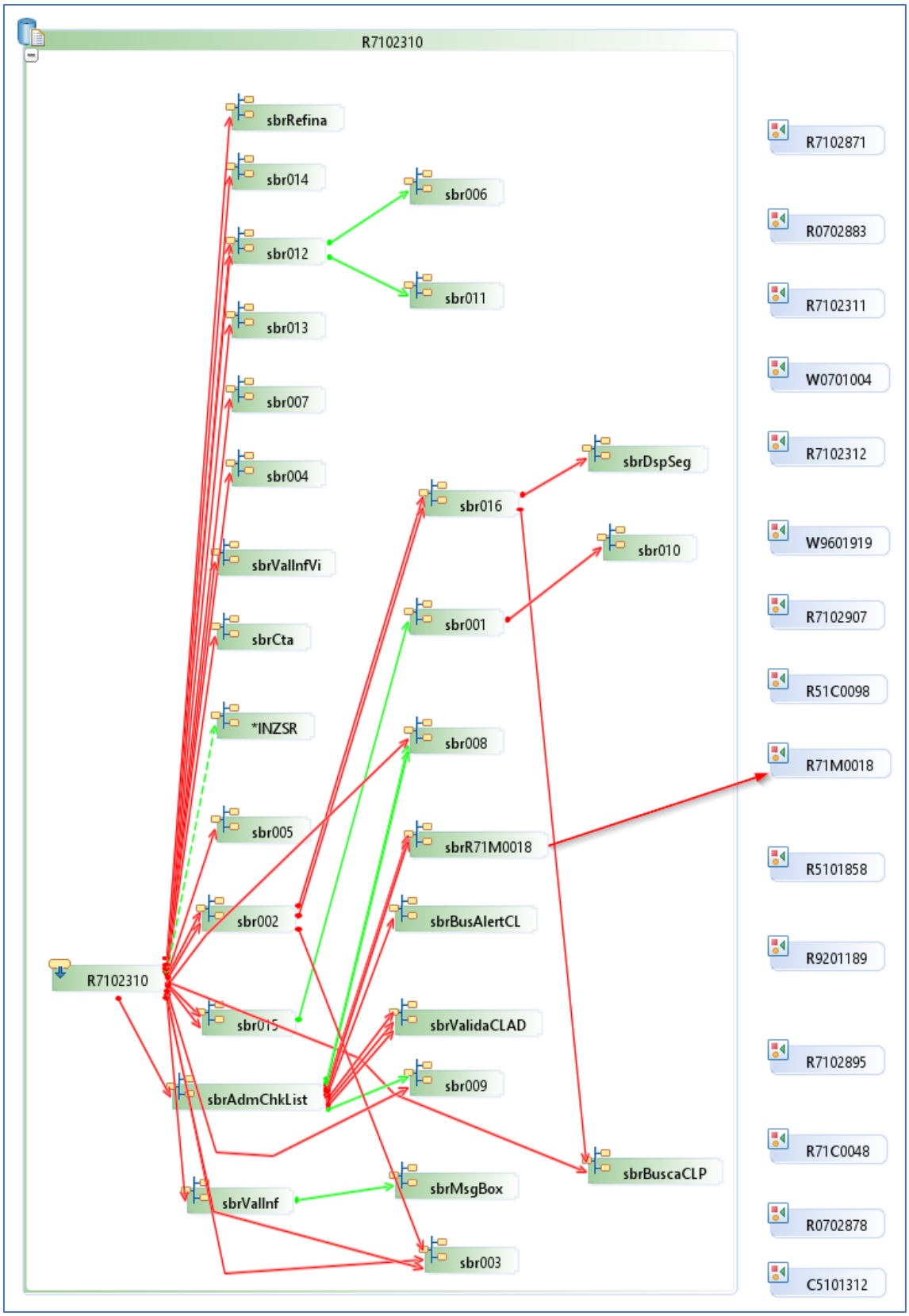


Figura 4.2.1.5. Diagrama del flujo de llamadas del programa R7102310.

La figura 4.2.1.5 demuestra gráficamente los síntomas de un mal diseño, que se exponen en el capítulo 3, con las cuales se ha construido el programa, y vamos a detallar cada uno de ellos contrastándolos con el programa **R7102310** que estamos exponiendo.

- **Rigidez**: La aplicación es difícil de comprender para poder agregar los cambios necesarios que se solicitaba, fue una tarea ardua encontrar los puntos exactos dónde aplicarlos sin afectar las funcionalidades existentes, el problema es que el programa tiene muchas responsabilidades y no aplica ninguna técnica de desarrollo de software que permita hacer una mejor abstracción de las reglas del negocio, por la cual cada cambio realizado tiene que ser hecho con extremo cuidado, evitando desencadenar cambios no deseados que ocasionen estragos al negocio de la empresa.

```
c                   eval      wservi1=SERVIC
c*                  eval      wservi1=80
c                   eval      sTipDoc=''
c                   eval      sDocume=''
c                   eval      *in41='0'
c                   eval      perr='0'
C                   move      psec          wsec
C                   move      pfec          wfec
C                   move      pemp          wemp
c                   Eval      nDia6=wDia
c                   Eval      nMes6=wMes
c                   Eval      nAno6=%Dec(%subst(pfec:7:2):2:0)
C                   move      pofic         wofic
C                   z-add     10            valsiz          3 0
C                   setoff                                    9091
C                   seton                                     7266
 *-Inicio del Ciclo
c                   dow       *in03='0'                                I----
c* Agregado por NEBAR - CHECK LIST
c                   exsr      sbrBuscaCLP
c* Fin agregado por NEBAR
c  n03              write     f00
c  n03              exfmt     contro01
c   62              eval      *in03=*off
c   62              eval      *in41=*off
c   62              eval      *in62=*off
 *-
c                   if        *in04='1'                                I----
c                   select                                             I----
c                   when      FIELD='WCODCLI' and *in98='0'            X...
c                   setoff                                    9091
c                   move      xfec6         pfec6
```

Figura 4.2.1.6. Cambio introducido al programa R7102310 para el Check-List de Desembolso.

La figura 4.2.1.6 muestra el cambio introducido enmarcado en un rectángulo rojo, en este caso es una subrutina que va a manejar la lógica de negocio relacionada al Check-List previo al desembolso para no afectar con la existente. En la figura se puede notar lo difícil que es comprender a simple vista el código fuente lo que dificulta tremendamente las tareas de mantenimiento.

- **Fragilidad**: Debido a su arquitectura monolítica, el programa **R7102310** tiene en su diseño el peligro de que cuando se introduce un solo cambio, este puede quebrar o afectar a la aplicación en muchos puntos. Por lo tanto, la introducción de un nuevo cambio tiene que ser aplicado en muchos puntos de la aplicación con la finalidad de

salvaguardar su integridad, cuidando que esta funcione correctamente con el cambio aplicado. En la figura 4.2.1.7 muestra los diferentes puntos dónde se han realizado los cambios al programa **R7102310** con la finalidad de que el funcionamiento anterior y el nuevo puedan coexistir sin colisionar el uno con el otro.

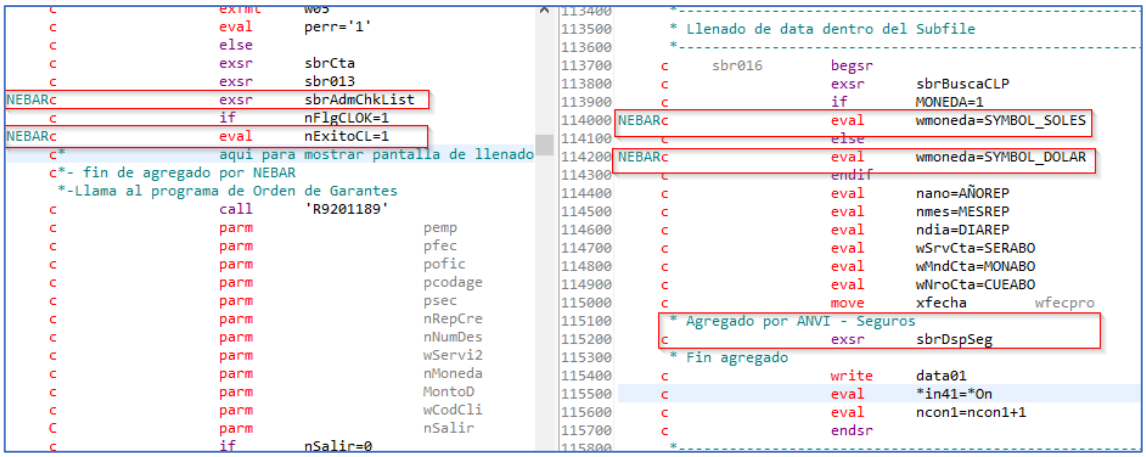


Figura 4.2.1.7. Cambio introducido al programa R7102310 para el Check-List de Desembolso.

- **Inmovilidad**: El programa **R7102310** contiene partes que pueden ser reutilizadas, sin embargo, es el esfuerzo que demanda separarlas del diseño inicial es demasiado grande por lo enmarañado de su construcción. Esto va a ser demostrado en la siguiente figura, la cual ha sido extraída con el uso de una herramienta IDE para hacer notar lo mencionado.

F

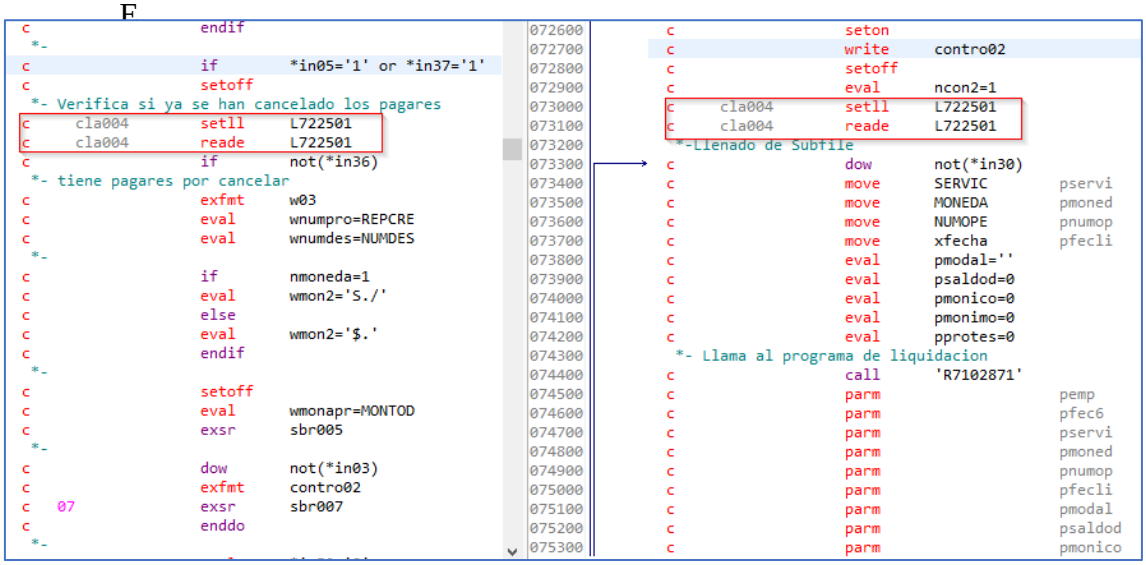


Figura 4.2.1.8. Fragilidad en el diseño del programa R7102310.

ura anterior, figura 4.2.1.8, resaltados con rectángulos de color rojo se exponen partes del programa **R7102310** que podrían haberse extraído, sin embargo, el esfuerzo de hacer todo ello hubiese requerido de la solicitud de un nuevo proyecto para abordarlo. Lo que hacen ambas porciones de código es validar si un pagaré de crédito ha sido cancelado, sin embargo, las mismas sentencias son repetidas en diferentes partes del programa entremezclada con otra lógica de negocio. No sólo se nota la viscosidad del diseño del programa, también quedan expuestas muchas malas prácticas de codificación y el uso de palabras muy antiguas del lenguaje RPG que la fecha en que fue construido resultan casi obsoletas.

- **Repetición innecesaria:** El diseño del programa **R7102310** contiene muchas estructuras que pueden ser agrupadas bajo un solo componente, lo cual podrían ayudar a tener una mejor comprensión del problema que resuelve o intuir de antemano acerca de qué trata. Especialmente

muchas operaciones de acceso a archivos de base de datos, las cuales pueden ser llevadas a un componente que evite la repetición de las mismas sentencias de código para lectura de un registro de un archivo de base de datos, la siguiente figura 4.2.1.9 muestra una porción de código que hace la misma operación con archivo de base de datos, una lectura de un registro en la tabla del maestro de clientes, en diferentes puntos de programa para luego proceder a realizar algunas evaluaciones basados en el resultado de la operación anterior.

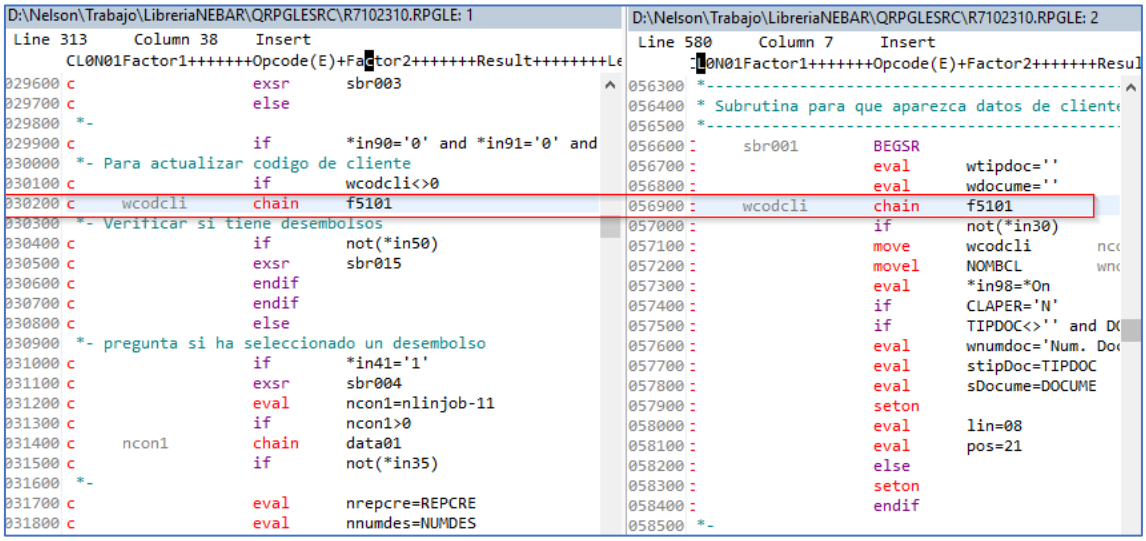
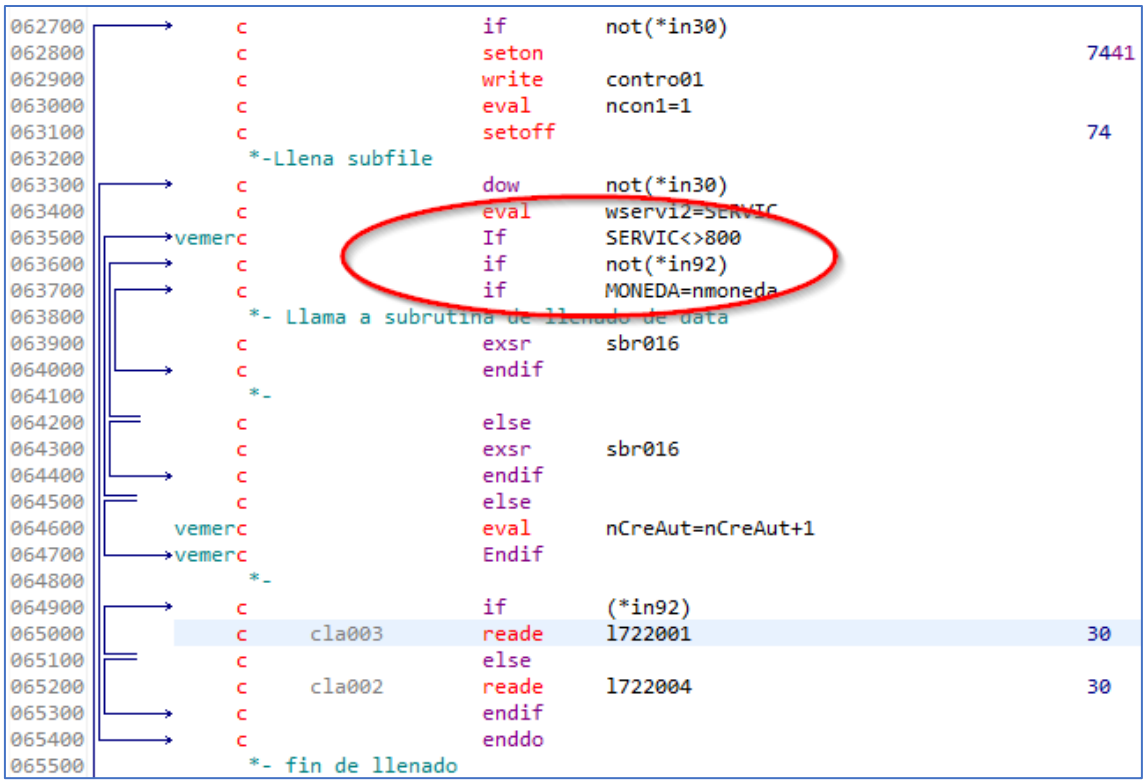


Figura 4.2.1.9. Repetición innecesaria de algunas características del programa R7102310.

- **Opacidad:** Este síntoma de mal diseño afirma que el diseño y el código son difíciles de leer. Estos no expresan su propósito correctamente, esta característica queda plasmada en gran medida en el programa R7102310, mucho de su código es ilegible, demasiado difícil de comprender, se ha tenido que acudir a labores de depuración para poder comprender la lógica del programa, y de esa forma introducir los cambios solicitados, con la finalidad de ubicar los puntos exactos dónde introducirlos sin alterar el flujo existente o producir comportamientos

extraños como resultado de la modificación hecha. La figura 4.2.1.10 muestra muchas malas prácticas de codificación en el programa **R7102310**, entre ellas tenemos el uso de números mágicos en el código del programa, es decir el uso de directo de un valor numérico, lo cual no dice nada acerca de lo que se trata, otra mala práctica expresada aquí es el uso de demasiadas sentencias "if" anidadas. Esto hace difícil comprender el diseño y la lectura de su código fuente.



```
062700      c                   if        not(*in30)
062800      c                   seton                                        7441
062900      c                   write     contro01
063000      c                   eval      ncon1=1
063100      c                   setoff                                       74
063200       *-Llena subfile
063300      c                   dow       not(*in30)
063400      c                   eval      wservi2=SERVIC
063500     vemerc               If        SERVIC<>800
063600      c                   if        not(*in92)
063700      c                   if        MONEDA=nmoneda
063800       *- Llama a subrutina de llenado de data
063900      c                   exsr      sbr016
064000      c                   endif
064100       *-
064200      c                   else
064300      c                   exsr      sbr016
064400      c                   endif
064500      c                   else
064600     vemerc               eval      nCreAut=nCreAut+1
064700     vemerc               Endif
064800       *-
064900      c                   if        (*in92)
065000      c       cla003      reade     l722001                            30
065100      c                   else
065200      c       cla002      reade     l722004                            30
065300      c                   endif
065400      c                   enddo
065500       *- fin de llenado
```

Figura 4.2.1.10. Opacidad del programa R7102310.

- **Programa R7101504:** Este es otro de los programas que se modificaron para adecuar el requerimiento de **Check-List**, y en su totalidad es una pequeña aplicación para consultar el detalle de los documentos de propuestas de desembolso para créditos de Empresa Financiera. A continuación, la figura 4.2.1.11 muestra en tiempo de ejecución como luce de cara al usuario, se ha señalado con un ovalo de color rojo la nueva opción a la pantalla del programa que se agregó como consecuencia del requerimiento. Dicha opción va a permitir hacer consultas del estado en que se encuentran cada uno de los tipos de **Check-List** que tenga asociados la propuesta de desembolso, es decir, el Check-List previo al desembolso y durante el desembolso, este último si la propuesta de desembolso ya se ha realizado.

Figura 4.2.1.11. Ejecución del programa R7101504, consultas de propuestas de desembolso.

Al igual que el programa **R7102310**, este posee las mismas características de diseño arquitectónico en su construcción, es decir un patrón arquitectónico monolítico, no muy útil para poder escalar la aplicación **(Bass, Clements, & Kazman, 2013, p. 6)**, no hay separación de responsabilidades en las tareas que lleva a cabo, además de exponer en su codificación muchas violaciones a los estándares que la industria del desarrollo de software ha establecido como buenas prácticas.

Para introducir los cambios necesarios para satisfacer las necesidades requerimiento del Check-List, fue necesario llevar a cabo un minucioso análisis con la finalidad de encontrar el punto exacto dónde introducirlos sin afectar la funcionalidad existente del programa. Para ilustrar esta gran dificultad, la figura 4.2.1.12 muestra el diagrama de componentes de este programa y la interacción

de las distintas subrutinas de las cuales está compuesto. Además, se ha resaltado con color rojo las flechas que ilustran de forma gráfica los cambios aplicados. Es notorio a simple vista lo enmarañado que es el diseño del programa, con el cual se ha tenido que lidiar para poder adaptar e introducir la nueva funcionalidad.

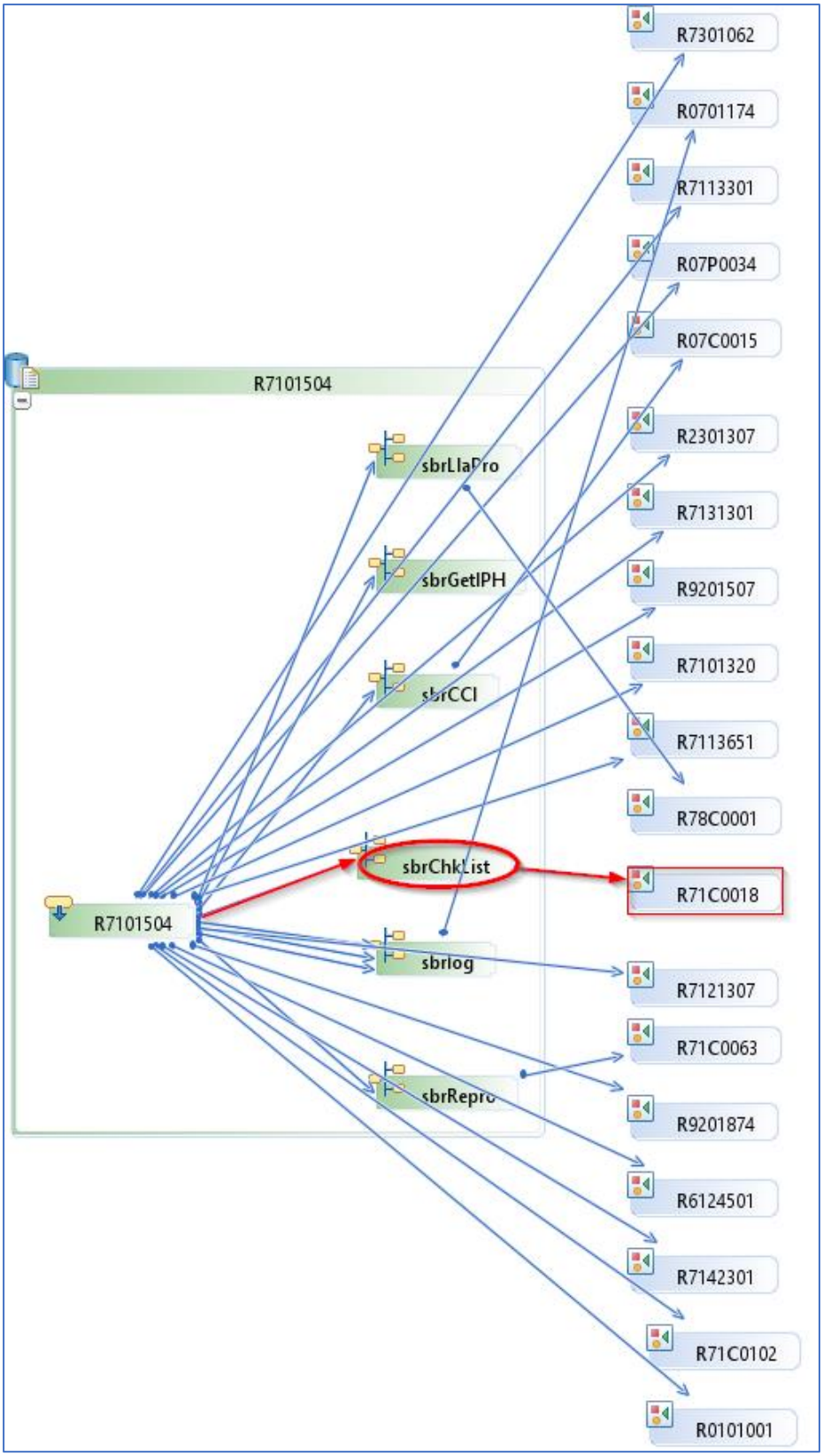


Figura 4.2.1.12. Diagrama de componente del programa R7101504 y la interacción de sus subrtutinas.

Además de mostrar las características que se acaban de mencionar, también cae en las malas prácticas que el programa **R7102310** posee en su diseño, vamos a señalar cada una de ellas exponiendo un extracto de su código por cada una de ellas, sin caer en la definición pues estas ya están dadas en la exposición del análisis del programa anterior y en el capítulo 3 se pueden consultar.

- **Rigidez**: La aplicación es muy difícil de cambiar sin generar daños colaterales con los cambios introducidos en ella y en un principio fue casi imposible estimar el tiempo para adecuar la aplicación al requerimiento **Check-List**, la figura 4.2.1.13 muestra un extracto de código del programa **R7101504** en el cual se puede notar la dificultad para leerlo y comprenderlo, sobre todo que está compuesto de un solo procedimiento en el cual está abstraída toda la lógica del negocio, es decir un diseño monolítico.

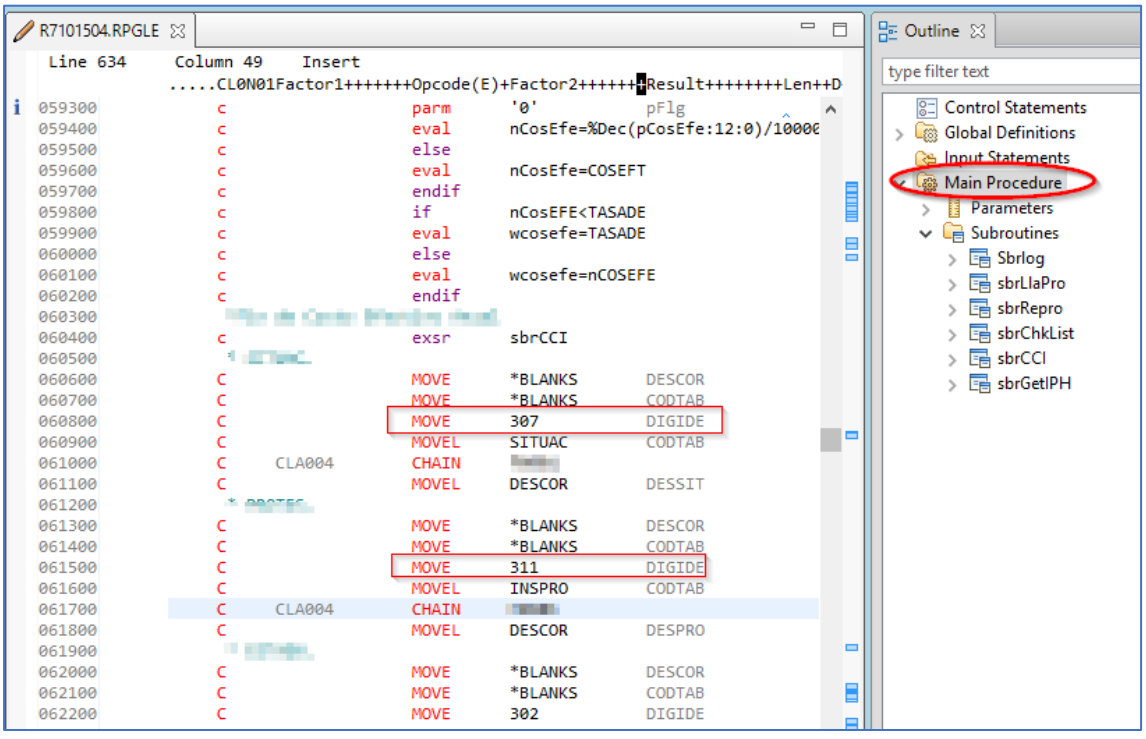


Figura 4.2.1.13. Diagrama de componente del programa R7101504 y la interacción de sus subrtutinas.

- **Fragilidad:** Los cambios introducidos por la adecuación del requerimiento de **Check-List** podrían haber causado que el sistema rompa muchas de sus funcionalidades que ya tenía. Esto sucede por qué el programa **R7101504** está diseñado usando subrutinas, cuyas variables son de ámbito global, quiere decir que están disponibles en el todo el programa, esto es un escenario peligroso ya que un cambio en alguna de ellas puede causar el colapso total de la aplicación. Es así como adecuar la aplicación representó un gran esfuerzo, para lograr esto se definió una nueva subrutina llamada ***sbrChkList*** que manejase la lógica de negocio para el requerimiento de ***Check-List***, modificar en la medida de lo posible el estado de las variables existentes, definir nuevas para evitar errores inesperados en la aplicación. La figura 4.2.1.14 muestra los cambios realizados para evitar en la medida de lo posible causar comportamientos o errores inesperados en la aplicación, los rectángulos rojos son los cambios introducidos para adecuar la aplicación.

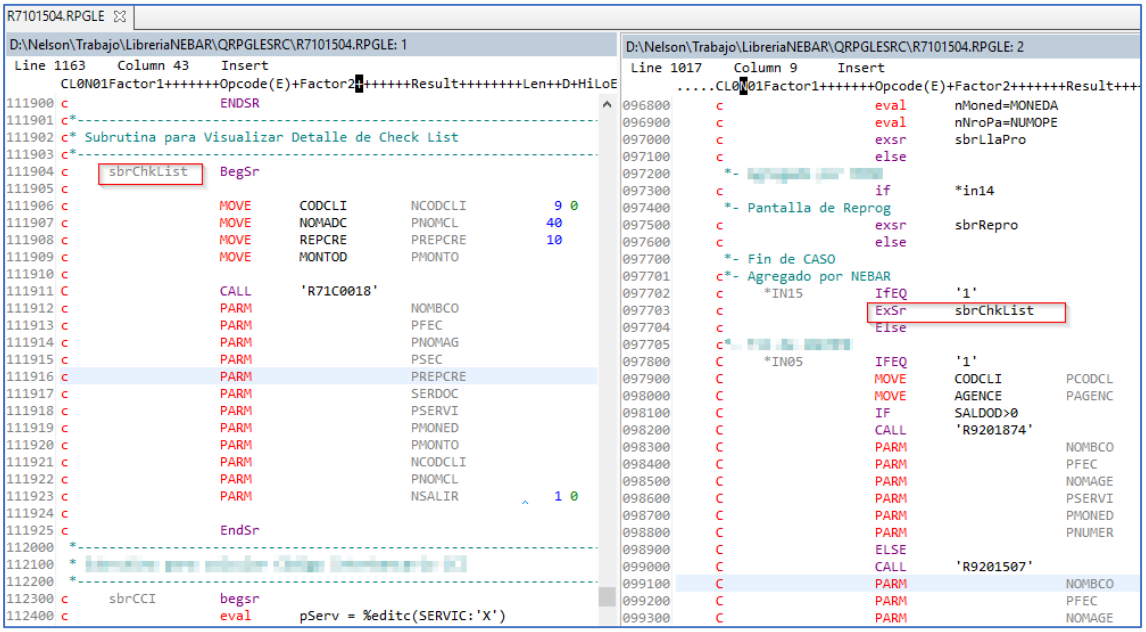


Figura 4.2.1.14. Rigidez en el programa R7101504.

- **Inmovilidad:** El programa R7101504 contiene muchas partes que pueden ser reusadas, pero el esfuerzo de extraerlas resulta muy grande y sobre todo representa un costo elevado, su diseño es demasiado enredado para poder extraer aquellas partes de código que podrían haber sido aprovechadas. La figura 4.2.1.15 muestra este mal síntoma en su diseño, en este caso concreto, repetidos acceso a una tabla de base de datos sistema para obtener un registro de esta, dicha operación pudo haber sido implementada en un procedimiento para ser reutilizado allí dónde es requerido, sin embargo, esta operación está esparcida en muchas partes del programa lo cual representa un gran esfuerzo rastrear todos los usos concretos que se ha hecho de este tipo de operación.

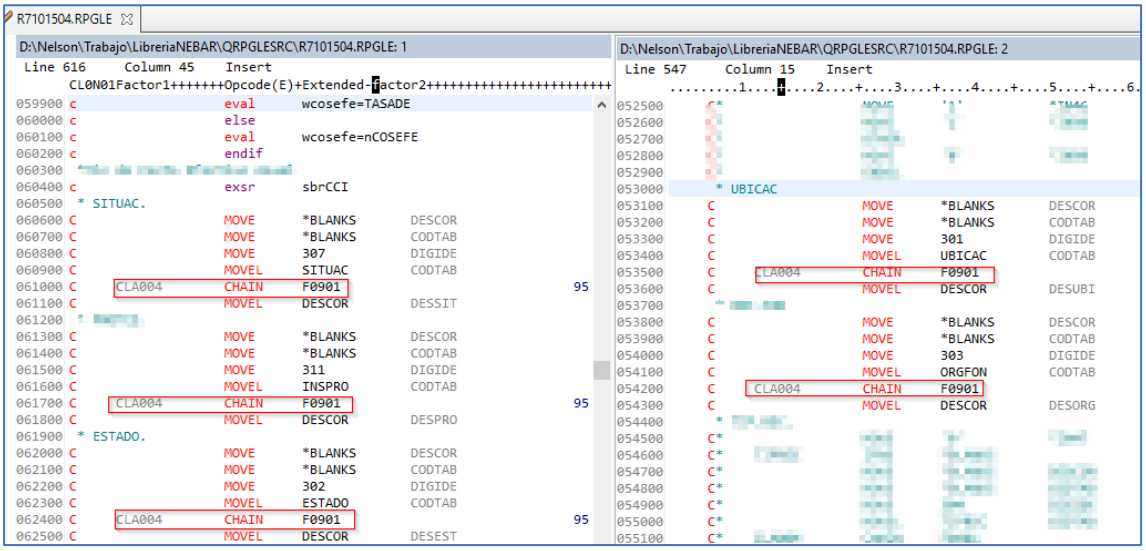


Figura 4.2.1.15. Fragilidad del programa R7101504 en implementación de la subrutina sbrChkList.

- **Repetición innecesaria:** El programa **R7101504** contiene muchas reglas de negocio en su implementación que puede ser agrupadas en un componente independiente para ser reutilizadas por muchos otros programas, sin embargo, estas se encuentran en un solo y gran componente, ya que como se ha mostrado este está compuesto de un solo procedimiento principal; pues el uso de subrutinas, que son los componentes más pequeños de los cuales está construido, son solo para uso local del programa y no cuentan con el uso independiente de variables locales, ni tampoco pueden ser invocadas externamente por otros programas. La figura 4.2.1.16 muestra el uso de este mal diseño, este pequeño trozo de código, cuyas partes resaltantes enmarcadas en rectángulos de color rojo, muestra reglas de negocio que son generales para todos los créditos que maneja la institución financiera y puede ser ubicada en un componente independiente para ser invocado allí dónde se le necesite evitando de la repetición innecesaria en cada programa.

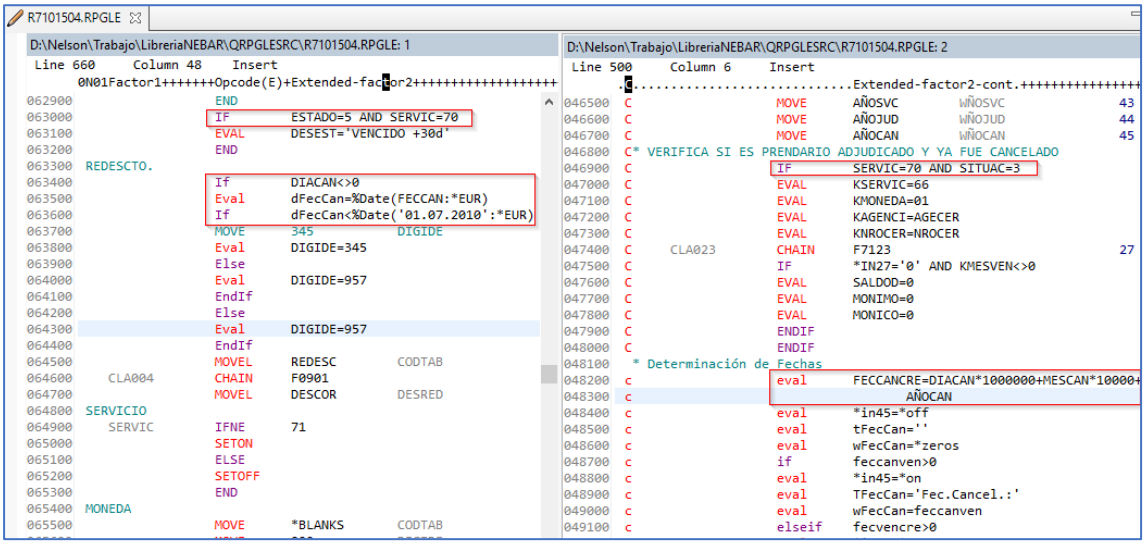
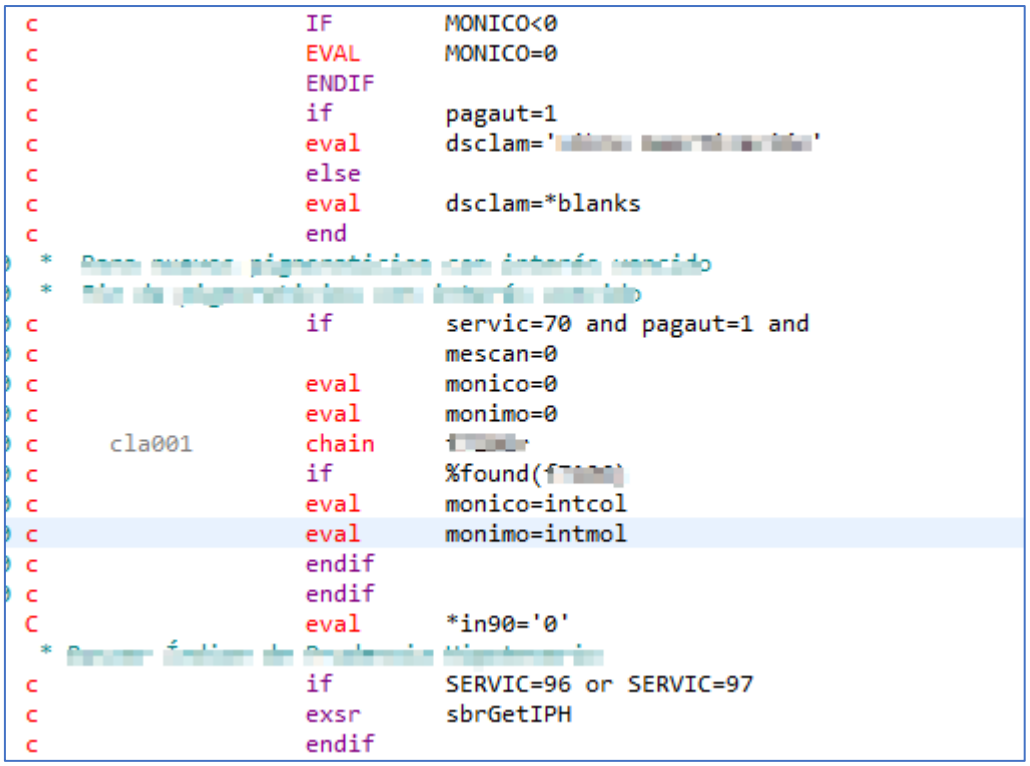


Figura 4.2.1.16. Inmovilidad en el diseño del programa R7101504.

- **Opacidad:** El estilo de codificación del programa **R7101504** es RPG encolumnado, cuyas reglas son muy rígidas en lo relacionado a la posición de las sentencias dentro del archivo de código fuente, sin embargo, este no tiene o expresa el propósito con claridad para el cual fue construido, tampoco es fácil de leer y comprender; muchas malas prácticas de codificación se ven expresadas a lo largo de todo el programa, no cumple la recomendación dada por Martin Fowler en la cual afirma que un buen código debe de comunicar claramente qué es lo que hace, y una manera de hacer esto es a través de la nomenclatura de variables **(Fowler, Beck, Brant, Opdyke, & Roberts, 1999, p. 22)**, sin embargo, esto no se aprecia en casi la totalidad del código del programa, a excepción del código agregado para la adecuación del requerimiento de Check-List. La figura 4.2.1.17 muestra con claridad lo que se acaba de mencionar, esta expone un extracto del código fuente del programa.



```
c                   IF        MONICO<0
c                   EVAL      MONICO=0
c                   ENDIF
c                   if        pagaut=1
c                   eval      dsclam='
c                   else
c                   eval      dsclam=*blanks
c                   end
 *
 *
c                   if        servic=70 and pagaut=1 and
c                             mescan=0
c                   eval      monico=0
c                   eval      monimo=0
c     cla001        chain
c                   if        %found(
c                   eval      monico=intcol
c                   eval      monimo=intmol
c                   endif
c                   endif
C                   eval      *in90='0'
 *
c                   if        SERVIC=96 or SERVIC=97
c                   exsr      sbrGetIPH
c                   endif
```

Figura 4.2.1.17. Opacidad del programa R7101504.

4.2.2 Desarrollo del nuevo marco de trabajo para el módulo de Créditos de una empresa financiera.

En el capítulo anterior se han expuesto todas las desventajas y falencias que representa construir aplicaciones empresariales usando una arquitectura monolítica con el lenguaje RPG, sobre la cual se basa el diseño de muchas aplicaciones de una EMPRESA FINANCIERA, es por ello que se ha tomado un enfoque diferente para que este nuevo marco de trabajo, se ha hecho uso de las técnicas modernas de la ingeniería de software, como son los patrones de diseño arquitectónicos, adaptándolos al paradigma de programación estructurado, ya que en su mayoría, por no decir todos, están enfocados ser usados bajo el paradigma de la programación orientada a objetos.

Este nuevo marco de trabajo ha tomado dos patrones arquitectónicos para su desarrollo, estos son: el patrón arquitectónico Modelo Vista Controlador y Patrón arquitectónico Orientado A Servicios SOA, cuyos conceptos ya se han expuesto en el capítulo 2, y de la característica del lenguaje de programación RPG denominada ILE, cuyo concepto se expone en el capítulo 3. A estos patrones arquitectónicos se les han hecho las adecuaciones necesarias para adaptarlo a la situación que se aborda en el presente trabajo y poder acoplarlos a la arquitectura ya existente. Juntando estos dos elementos importantes, tanto el que ofrece la ingeniería de software y la tecnología IBM (entorno de desarrollo ILE para RPG) se ha desarrollado este nuevo marco de trabajo para dar solución a este nuevo requerimiento de software para el módulo de créditos de Empresa Financiera, el cual encaja de forma perfecta para abordarlo y mostrar la manera de cómo aplicarlo.

4.2.2.1 Base de datos del requerimiento de check-list para el módulo de créditos de una empresa financiera.

Uno de los componentes de toda aplicación de negocio es la base de datos, en este componente va a ser persistida toda la información necesaria e importante para el correcto funcionamiento de esta y del negocio. Por otro lado, la base de datos puede ser considerado un componente arquitectónico **(Fowler, Design - Who needs an architect?, 2002, pág. 3)**, si se tiene en cuenta la restricción que esta no se puede cambiar, y se está obligado a usar el motor de base de datos con el cual el usuario cuenta.

Tomando este principio, el motor de base de datos que va a ser usado es ***DB2 for i***. Este motor de base datos viene integrado dentro del sistema Operativo ***IBM i*** con el servidor **POWER SYSTEM**, esto quiere decir que todo el acceso a

la base de datos es implementado a través del sistema operativo, incluso algunos componentes hardware son usado para lograr una mejor performance **(Bedoya, Cruz, Lema, & Singkorapoom, 2016, p. 2).** Esta integración con el sistema operativo representa una gran ventaja para procesar grandes volúmenes de transacciones que las empresas del sector financiero suelen requerir, es por ello, que la velocidad en el procesamiento de los datos es una restricción que tiene que ser tratado con sumo cuidado para satisfacer los requerimientos de calidad de una buena arquitectura de software. Una mayor discusión acerca de la base datos ***DB2 for i*** esta fuera del alcance del presente trabajo.

Lo siguiente a realizar es crear el modelo entidad relación que permita plasmar las nuevas entidades de negocio para el requerimiento Check-List, y que está pueda integrarse con la base de datos ya existente del sistema de gestión de créditos de una Empresa Financiera. Se va a describir las nuevas tablas creadas y, se va a mencionar algunas de las tablas más importantes de esta parte del negocio o también llamado módulo que compone el aplicativo **SOFIA** con la finalidad de poder exponer con claridad el contexto expuesto en este trabajo, pues por motivos de confidencialidad, no es posible cubrir a detalle la base de datos en su totalidad. La figura 4.2.2.1.1 muestra parte del modelo entidad relación del modelo de base de datos para la aplicación del marco de trabajo propuesto, sin embargo, este modelo sí contiene todas las nuevas entidades creadas, pero no todas aquellas con las que ya cuenta el módulo de gestión de créditos de una Empresa Financiera denominado **SGCRED** (Sistema de gestión de créditos).

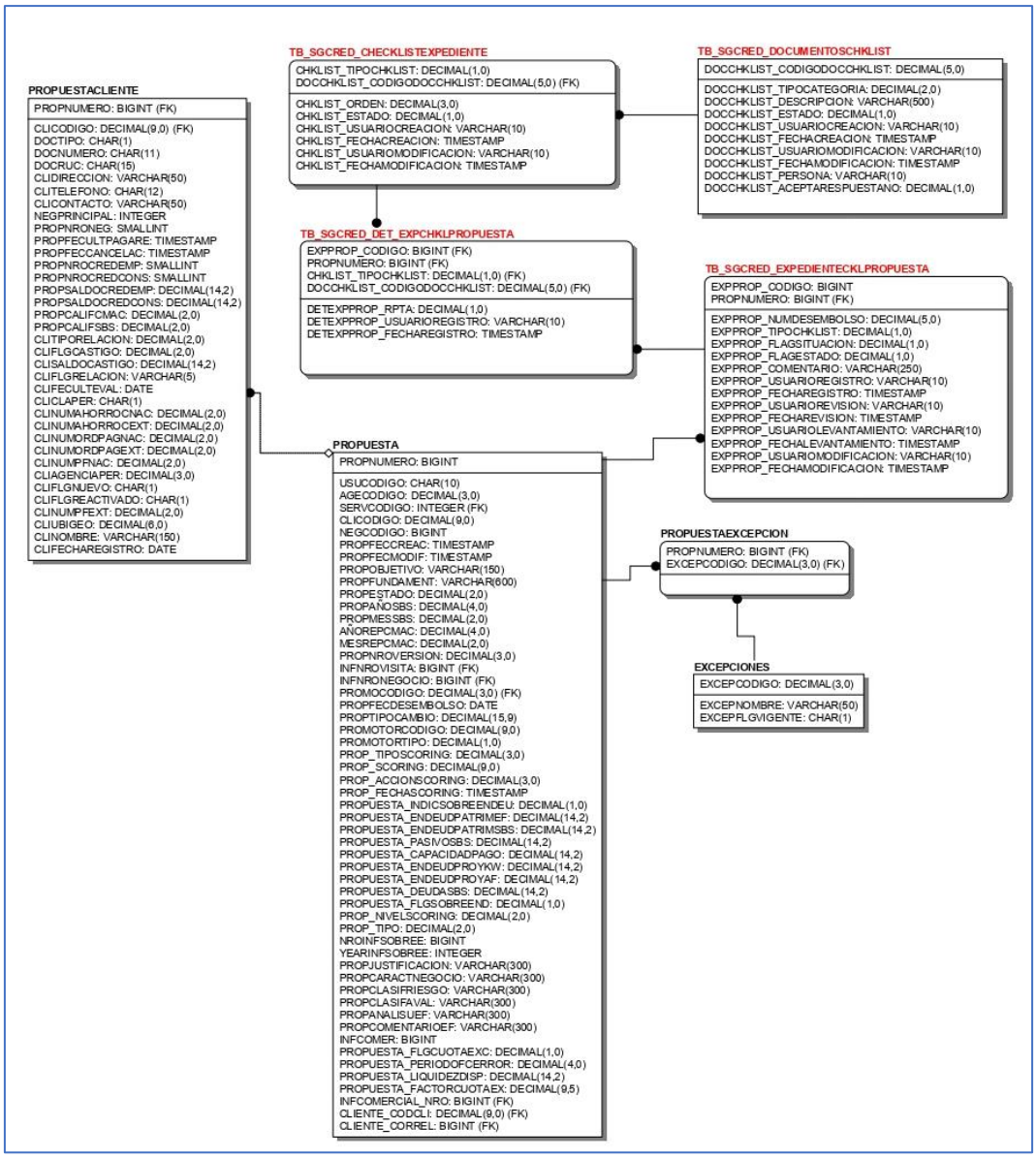


Figura 4.2.2.1.1.Modelo Entidad Relación para el requerimiento Check-List.

El modelo entidad relación de la base de datos para el Check-List cuenta con cuatro nuevas tablas necesarias para satisfacer las necesidades de esta nueva funcionalidad, las del Check-List previo y antes durante un desembolso en una Empresa Financiera, estas a su vez se integran con la base de datos existentes. En este modelo podemos apreciar solo unas cuantas, debido a la restricción de

confidencialidad de datos de la entidad financiera, y se han mostrado sólo las necesarias para dar claridad al trabajo que se expone como ya se ha mencionado. Los nombres de las tablas que muestra el modelo son los siguientes:

- Tabla **TB_SGCRED_CHECKLISTEXPEDIENTE**: Contiene información relacionada a los documentos por tipo de Check-List.
- Tabla **TB_SGCRED_DOCUMENTOSCHKLIST**: Contiene información los documentos por cada tipo de Check-List.
- Tabla **TB_SGCRED_EXPEDIENTECKLPROPUESTA**: Contiene información de los expedientes de Check-List por cada propuesta del cliente.
- Tabla **TB_SGCRED_DET_EXPCHKLPROPUESTA**: Contiene información de la relación de muchos a muchos entre los expedientes de Check-List y las propuestas de desembolso.
- Tabla **PROPUESTACLIENTE**: Contiene información de la propuesta de desembolso para un cliente de Empresa Financiera.
- Tabla **PROPUESTA**: Contiene información de los datos relacionados a las características de la propuesta, como son el tipo de producto a la cual está dicha propuesta.
- Tabla **EXCEPCIONES**: Contiene información relacionada a las excepciones asignadas a la propuesta de desembolso.
- Tabla **PROPUESTAEXCEPCION**: Contiene información de la relación de muchos a muchos entre las excepciones y las propuestas de desembolso.

  Las tablas 4.1 a la 4.4 muestran el diseño de cada las tablas que se acaban de exponer, considerando sólo aquellas que se han creado para

el requerimiento, pero no las que ya existían por las restricciones que se han mencionado.

Tabla 4.1.

Columnas de la tabla ***TB_SGCRED_CHECKLISTEXPEDIENTE***

Nombre Campo	Tipo de Dato	Descripcion
CHKLIST_TIPOCHKLIST	DECIMAL(1,0)	TIPO DE EXPEDIENTE CHECK LIST TABAL F6209 DIGIDE 627
DOCCHKLIST_CODIGODOCCHKLIST	DECIMAL(5,0)	CODIGO DE DOCUMENTO DE CHECK LIST
CHKLIST_ORDEN	DECIMAL(3,0)	ORDEN DE DOCUMENTO EN EL EXPEDIENTE
CHKLIST_ESTADO	DECIMAL(1,0)	ESTADO DE DOCUMENTO EN EL EXPEDIENTE. 0 : PENDIENTE. 1 : ACTIVO. 2: ELIMINADO.
CHKLIST_USUARIOCREACION	VARCHAR(10)	USUARIO DE CREACION DE DOCUMENTO EN EL EXPEDIENTE
CHKLIST_FECHACREACION	TIMESTAMP	FECHA DE CREACION DE DOCUMENTO EN EL EXPEDIENTE
CHKLIST_USUARIOMODIFICACION	VARCHAR(10)	USUARIO DE MOFICACION DE DOCUMENTO EN EL EXPEDIENTE
CHKLIST_FECHAMODIFICACION	TIMESTAMP	FECHA DE MODIFICACON DE DOCUMENTO EN EL EXPEDIENTE

Diseño de la tabla para los datos del expediente de Check-List.

Tabla 4.2.

Columnas de la tabla **TB_SGCRED_DOCUMENTOSCHKLIST**

Nombre Campo	Tipo de Dato	Descripcion
DOCCHKLIST_CODIGODOCCHKLIST	DECIMAL(5,0)	CODIGO DE DOCUMENTO DE CHECK LIST
DOCCHKLIST_TIPOCATEGORIA	DECIMAL(2,0)	TIPO DE CATEGORIA DE CHECK LIST TABAL F6209 DIGIDE 626
DOCCHKLIST_DESCRIPCION	VARCHAR(500)	DESCRIPCIÓN DE CHECK LIST

DOCCHKLIST_ESTADO	DECIMAL(1,0)	ESTADO DE DOCUMENTO DE CHECK LIST 0:INACTIVO. 1: ACTIVO. 2: ELIMINADO
DOCCHKLIST_USUARIOCREACION	VARCHAR(10)	USUARIO DE CREACIÓN DE DOCUMENTO DE CHECK LIST
DOCCHKLIST_FECHACREACION	TIMESTAMP	FECHA DE CREACIÓN DE DOCUMENTO DE CHECK LIST
DOCCHKLIST_USUARIOMODIFICACION	VARCHAR(10)	USUARIO DE MODIFICACIÓN DE DOCUMENTO
DOCCHKLIST_FECHAMODIFICACION	TIMESTAMP	FECHA DE MODIFICACIÓN DE DOCUEMENTO
DOCCHKLIST_PERSONA	VARCHAR(10)	A QUE TIPO DE PERSONA APLICA LA PREGUNTA : F0901 DIGIDE 4

Diseño de la tabla para los datos de los documentos que posee un Check-List.

Tabla 4.3.

Columnas de la tabla **TB_SGCRED_EXPEDIENTECKLPROPUESTA**

Nombre Campo	Tipo de Dato	Descripcion
EXPPROP_CODIGO	BIGINT	CODIGO DE CHECK LIST PARA PROPUESTA
PROPNUMERO	BIGINT	
EXPPROP_NUMDESEMBOLSO	DECIMAL(5,0)	NUMERO DE DESEMBOLSO
EXPPROP_TIPOCHKLIST	DECIMAL(1,0)	TIPO DE EXPEDIENTE CHECK LIST TABAL F6209 DIGIDE 627
EXPPROP_FLAGSITUACION	DECIMAL(1,0)	FLAG DE SITUACION DE CHECK LIST 0: SIN ENVIAR 1: ENVIADO 2: DEVUELTO/PENDIENTE 3: APROBADO
EXPPROP_FLAGESTADO	DECIMAL(1,0)	FLAG DE ESTADO: 0:INACTIVO 1:ACTIVO 2: ANULADO
EXPPROP_COMENTARIO	VARCHAR(250)	COMENTARIO DE REVISIÓN DE SECTORISTA LIDER O APROBADOR
EXPPROP_USUARIOREGIS	VARCHAR(10)	USUARIO DE REGISTRO DE CHECK

TRO		LISTA PARA PROPUESTA
EXPPROP_FECHAREGISTRO	TIMESTAMP	FECHA DE REGISTRO DE CHECK LISTA PARA PROPUESTA
EXPPROP_USUARIOREVISION	VARCHAR(10)	USUARIO REVISIÓN DE EXPEDIENTE
EXPPROP_FECHAREVISION	TIMESTAMP	FECHA REVISIÓN DE EXPEDIENTE
EXPPROP_USUARIOLEVANTAMIENTO	VARCHAR(10)	USUARIO LEVANTAMIENTO DE RESTRICCIÓN DE CHECK LIST
EXPPROP_FECHALEVANTAMIENTO	TIMESTAMP	FECHA LEVANTAMIENTO DE RESTRICCIÓN DE CHEK LIST
EXPPROP_USUARIOMODIFICACION	VARCHAR(10)	USUARIO MODIFICACION
EXPPROP_FECHAMODIFICACION	TIMESTAMP	FECHA DE MODIFICACION

Diseño de la tabla para los datos de las propuestas de desembolso que tienen un expediente de Check-List.

Tabla 4.4.

Columnas de la tabla **TB_SGCRED_DET_EXPCHKLPROPUESTA**

Nombre Campo	Tipo de Dato	Descripcion
EXPPROP_CODIGO	BIGINT	CODIGO DE CHECK LIST PARA PROPUESTA
PROPNUMERO	BIGINT	
CHKLIST_TIPOCHKLIST	DECIMAL(1,0)	TIPO DE EXPEDIENTE CHECK LIST TABLA F6209 DIGIDE 627
DOCCHKLIST_CODIGODOCCHKLIST	DECIMAL(5,0)	CODIGO DE DOCUMENTO DE CHECK LIST
DETEXPPROP_RPTA	DECIMAL(1,0)	RESPUESTA REGISTRO DE EXPEDIENTE 0:=NULO 1:SGCRED 2:EXPEDIENTE 3:NO APLICA
DETEXPPROP_USUARIOREGISTRO	VARCHAR(10)	USUARIO DE REGISTRO DE EXPEDIENTE
DETEXPPROP_FECHAREGISTRO	TIMESTAMP	FECHA REGISTRO DE EXPEDIENTE

Diseño de la tabla para los datos de la relación de muchos entre los tipos de Check-List y las propuestas de desembolso.

4.2.2.2 Programa de servicio B07U0002: check-list propuesta de desembolso.

Este componente de software dentro del contexto de ILE y RPG es un objeto de tipo ***SRVPGM (Diedrich, y otros, 2016, pág. 78)** que contiene los servicios principales para interactuar con la base de datos, es decir, con el modelo de la aplicación creada para este nuevo requerimiento. El modelo y las reglas del negocio se plasman en un solo lugar, en este componente todas están implementadas, para lograr un mejor encapsulamiento del sistema y que les permite ser utilizadas en cualquier lugar donde se las necesite y no estar localizadas en el mismo archivo de código fuente mezcladas con la interacción de la interfaz de usuario u otro componente tal como se ha había hecho.

Cada programa de servicio esta compuesto de uno o más objetos tipo ***MODULE** los cuales contienen subprocedimientos **(Bedoya, Cruz, Lema, & Singkorapoom, 2016, p. 46)** que encapsulan alguna regla de negocio, se puede entender cada subprocedimiento como un método en un lenguaje de programación como lo es java, C++ u otro lenguaje de programación. Cómo se ha explicado en el capítulo 3 dedicado a este tipo de objetos, estos pueden contener variables locales lo cual da una gran ventaja a la hora de desarrollar componentes reutilizables **(Diedrich, y otros, 2016, pág. 78)**. Para el caso del programa de servicio al cual no estamos refiriendo, **B07U0002**, este consta de una solo modulo, **B07U0002**, cuya estructura se expone en la figura 4.2.2.2.1.

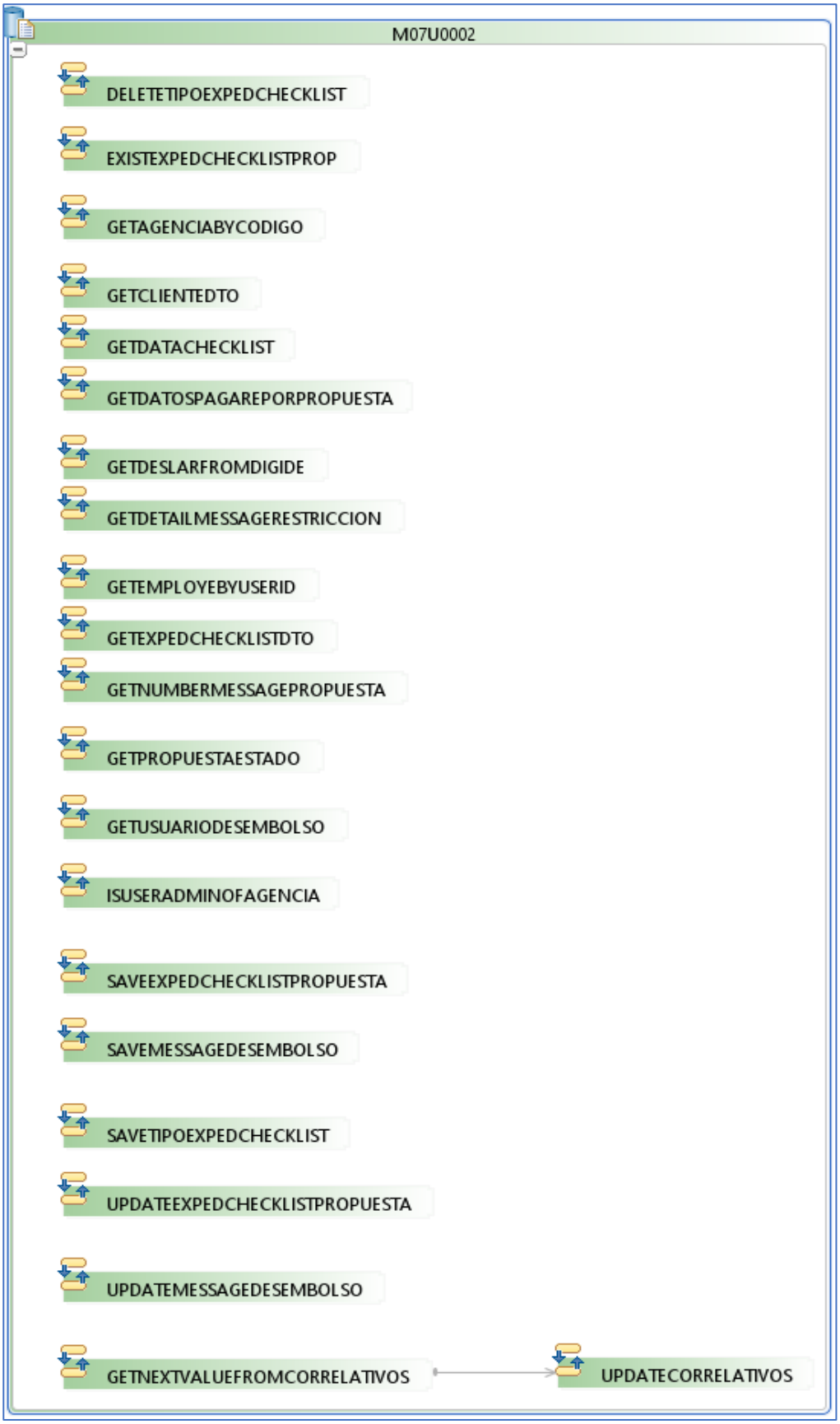


Figura 4.2.2.2.1. Diagrama de componente del módulo M07U0002.

La figura 4.2.2.2.1 muestra el módulo que compone el programa de servicio **B07U0002** con sus subprocedimientos del cual está compuesto, estos a su vez se convertirán en los servicios que serán ofrecidos a los clientes que deseen hacer consumo de ellos de acuerdo su necesidad. Esto se logra a través de la construcción del programa de servicio ejecutando el comando ***CRTSRVPGM*** **(Bedoya, Cruz, Lema, & Singkorapoom, 2016, p. 82)** el cual toma la declaración de cada uno de los procedimientos expresados en el lenguaje de enlazado, ***BINDER LANGUAGE*** **(Bedoya, Cruz, Lema, & Singkorapoom, 2016, p. 80)**, concepto explicado en el capítulo 3, cuyo código fuente ***B07U0002.BND*** se expone en la figura 4.2.2.2.2.

```
/* MODULO       : SISTEMA DE GESTION DE CREDITOS                    */
/* APLICACION   : CHECK LIST                                        */
/* ---------------------------------------------------------------- */
/* AUTOR        : Nelson Abel Barranzuela Iman                      */
/* FECHA CREAC. :                                                   */
/* ---------------------------------------------------------------- */
/* MODIFICADO POR :                          FECHA MODIF.:          */
/* ================                          =============          */
/*------------------------------------------------------------------*/
             STRPGMEXP  PGMLVL(*CURRENT) SIGNATURE('CHECKLITS 20150817 V1')
                EXPORT     SYMBOL(GETDATACHECKLIST)
                EXPORT     SYMBOL(GETNUMBERMESSAGEPROPUESTA)
                EXPORT     SYMBOL(SAVETIPOEXPEDCHECKLIST)
                EXPORT     SYMBOL(SAVEEXPEDCHECKLISTPROPUESTA)
                EXPORT     SYMBOL(UPDATEEXPEDCHECKLISTPROPUESTA)
                EXPORT     SYMBOL(DELETETIPOEXPEDCHECKLIST)
                EXPORT     SYMBOL(EXISTEXPEDCHECKLISTPROP)
                EXPORT     SYMBOL(GETEXPEDCHECKLISTDTO)
                EXPORT     SYMBOL(GETDESLARFROMDIGIDE)
                EXPORT     SYMBOL(GETNEXTVALUEFROMCORRELATIVOS)
                EXPORT     SYMBOL(UPDATECORRELATIVOS)
                EXPORT     SYMBOL(SAVEMESSAGEDESEMBOLSO)
                EXPORT     SYMBOL(UPDATEMESSAGEDESEMBOLSO)
                EXPORT     SYMBOL(GETCLIENTEDTO)
                EXPORT     SYMBOL(GETPROPUESTAESTADO)
                EXPORT     SYMBOL(GETDATOSPAGAREPORPROPUESTA)
                EXPORT     SYMBOL(GETUSUARIODESEMBOLSO)
                EXPORT     SYMBOL(GETDETAILMESSAGERESTRICCION)
                EXPORT     SYMBOL(GETEMPLOYEBYUSERID)
                EXPORT     SYMBOL(GETAGENCIABYCODIGO)
                EXPORT     SYMBOL(ISUSERADMINOFAGENCIA)
             ENDPGMEXP
```

Figura 4.2.2.2.2. Lenguaje de enlazado para el programa de servicio B07U0002.

El lenguaje de enlazado permite hacer público cada subprocedimiento que se encuentra en los módulos de los que está compuesto un programa de servicio, este se puede considerar como la interfaz que ofrece una API en la cual el cliente no sabe nada acerca de la implementación interna de esta, esto proporciona un buen encapsulamiento y representa una buena característica de diseño, permite crear una arquitectura basada en capas **(Amra, y otros, 2014, pág. 34)**. Finalmente, al crear el programa de servicio se obtiene un resultado como el que muestra el diagrama de componentes en la figura 4.2.2.2.3 .

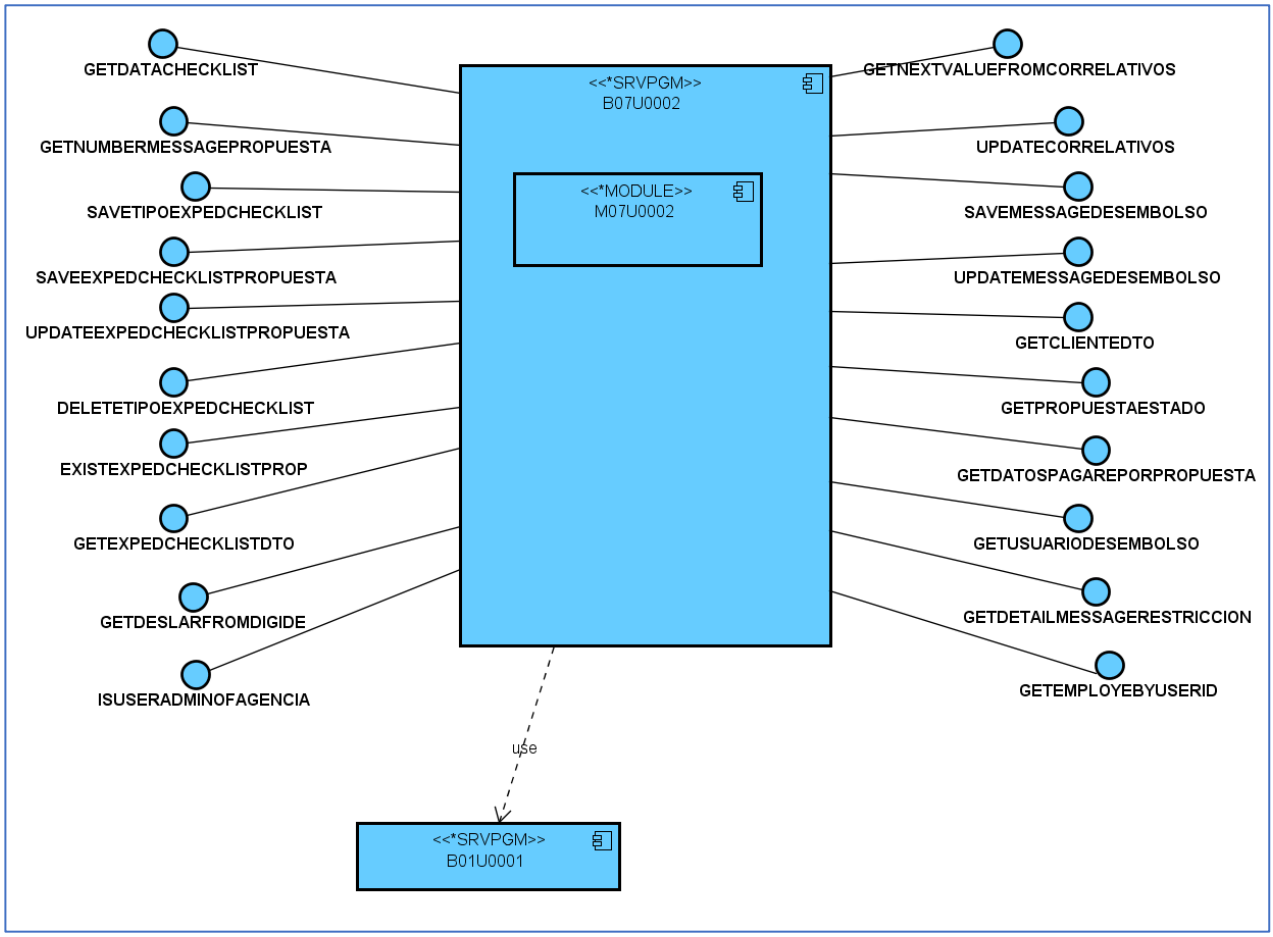


Figura 4.2.2.2.3. Diagrama de componentes del programa de Servicio B07U0002.

El diagrama de componentes del programa de servicio **B07U0002** de la figura 4.2.2.2.3 muestra los servicios que ofrece, el objeto módulo del cual está compuesto y el uso que hace de otro programa de servicio, en este caso **B07U001**, este último programa de servicio provee un conjunto utilidades que no tienen que ver con las reglas de negocio, más delante se va a exponer su estructura. El sistema operativo provee un comando para ver la estructura del objeto una vez que ha sido creado, este comando se ha expuesto en el capítulo 3, el comando es **DSPSRVPGM (Diedrich, y otros, 2016, pág. 143)** que muestra en consola la información relacionada al programa de servicio. Las siguientes ilustraciones, figuras 4.2.2.2.4 a la 4.2.2.2.6 muestran la salida en pantalla la ejecución del comando mencionado, las cuales son las más importantes ,ya que dicho comando arroja como resultado muchas pantallas.

Figura 4.2.2.2.4.Resultado de aplicar el comando DSPSRVPGM al programa B07U0002.

Figura 4.2.2.2.5.Resultado de aplicar el comando DSPSRVPGM al programa B07U0002, módulos que lo componen.

Figura 4.2.2.2.6. Resultado de aplicar el comando DSPSRVPGM al programa B07U0002, subprocedimientos exportados.

Finalmente, el programa de servicio debe de ser instalado en un lugar estándar para que los programas clientes puedan consumirlos, es decir enlazarlos (***binding***), este lugar se llama directorio de enlace **BINDING DIRECTORY**

(Diedrich, y otros, 2016, pág. 44), en el cual se instala el programa de servicio que hemos creado, así como otros que la aplicación pudiese requerir. Para los programas de servicio que se exponen en el presente trabajo, se ha usado un solo directorio de enlace denominado: **SRVPBND**, este nombre es muy importante para tener en cuenta, ya que es la manera como los programas clientes van a poder hacer referencia a los programas de servicio que necesiten consumir. Hecha la instalación se puede consultar el directorio de enlace para poder visualizar las entradas que contiene usando el comando **WRKBNDDIRE (Diedrich, y otros, 2016, pág. 79)**, la figura 4.2.2.2.7 muestra el resultado de la ejecución del comando mencionado.

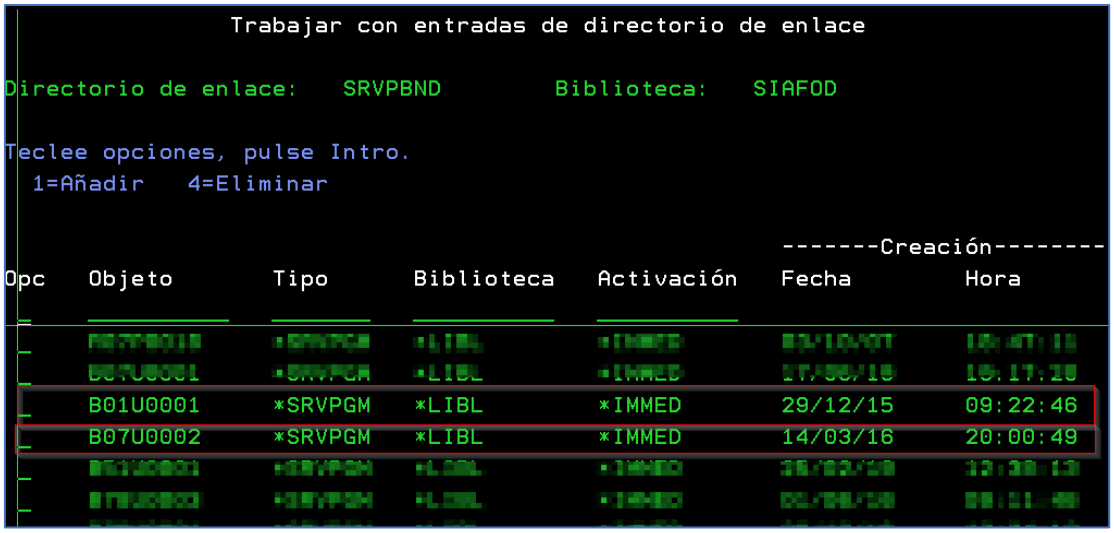


Figura 4.2.2.2.7. Resultado de la ejecución del comando WRKBNDDIRE para el directorio de enlace SRVPBND.

4.2.2.3 Programa de servicio B07U0002: utilidades para el manejo de Errores y ejecución de algunos comandos del sistema operativo.

Este componente, al igual que el componente **B07U0002** en un objeto de tipo ***SRVPGM (Diedrich, y otros, 2016, pág. 78)**, cuyo concepto se explica en el capítulo 3 del presente trabajo. En este se han implementado aquellos aspectos de la aplicación que no tienen relación conceptual con el dominio de la

aplicación, sino que, por el contrario, son aspectos relacionados con el manejo de errores de la aplicación e interacciones con funcionalidades que ofrece el sistema operativo ***i-system***, como resultado de la ejecución de esta.

Con la construcción de este programa de servicio se busca aplicar el principio de diseño de software ***single responsibility*** **(Amra, y otros, 2014, pág. 33)**, es decir única responsabilidad por cada componente, con la finalidad de atacar el problema al que estamos tratando de resolver, pues como se ha expuesto, la arquitectura monolítica no permite la separación de responsabilidades, y en la construcción de la aplicación con la cual se ha estado trabajando se mezcla todo en un solo componente, incluso el manejo de aquellas tareas de interacción con las APIS de bajo de nivel que ofrece el sistema operativo ***i-system*** para extender la funcionalidad de la aplicación.

Como ya se ha mencionado, cada programa de servicio está compuesto por módulos, objetos tipo ***MODULE (Diedrich, y otros, 2016, pág. 78)**; cuyo concepto se explica en el capítulo 3, en el cual a su vez está compuesto de subprocedimientos **(Diedrich, y otros, 2016, págs. 46-48)** que luego serán exportados o hechos públicos a través de la creación del programa de servicio. El programa de servicio **B07U0001** está compuesto de un solo modulo, el módulo **M01U0001**, la figura 4.2.2.3.1 muestra su respectivo diagrama de componente.

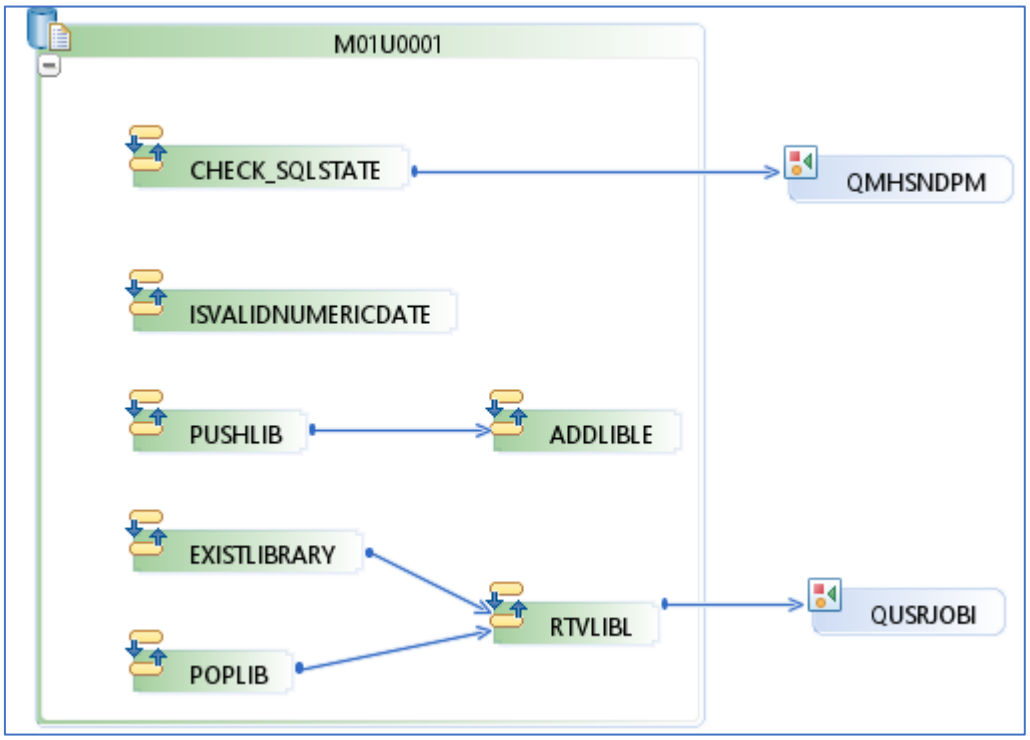


Figura 4.2.2.3.1. Diagrama de Componente el módulo M01U0001.

Otros de los elementos de los cuales son necesario para la construcción de un programa de servicio, es el lenguaje de enlazado: ***BINDER LANGUAGE***, en este se expresan dos cosas importantes, la primera es aquí donde se declaran cada uno de los procedimientos que van a ser exportados desde el módulo a través de la creación del programa de servicio **(Diedrich, y otros, 2016, pág. 80)**, y la segunda es la marca o firma con la cual se construye un programa de servicio, para este caso se ha creado la firma: "**23102015 V1**" a través de la palabra clave del lenguaje de enlazado ***SIGNATURE***. La marca o firma de un programa de servicio es importante, es con la cual los programas clientes lo enlazan, y si está cambia ocasiona errores en tiempo de ejecución en los programas clientes que lo consumen **(Diedrich, y otros, 2016, pág. 80)**. La

figura 4.2.2.3.2 muestra el código fuente del lenguaje de enlazado para el programa de servicio en el miembro de código fuente : ***B01U0001.BND***.

B01U0001.BND

```
 Line 29      Column 2     Insert
      -+---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000100 /*--------------------------------------------------------------------*/
000101 /*              SOFTWARE FINANCIERO AUTOMATIZADO (SOFIA)               */
000102 /*              ***************************************               */
000103 /* MODULO       : UNIDAD DE CUMPLIMIENTO                              */
000104 /* APLICACION   : UTILIDADES DE SISTEMAS                              */
000105 /*                                                                    */
000106 /* ------------------------------------------------------------------ */
000107 /* AUTOR        : Nelson Abel Barranzuela Iman                        */
000108 /* FECHA CREAC. :                                                     */
000109 /* ------------------------------------------------------------------ */
000110 /* MODIFICADO POR :                          FECHA MODIF.:            */
000111 /* ================                          =============            */
000112 /*                                                                    */
000113 /*--------------------------------------------------------------------*/
000115              STRPGMEXP  PGMLVL(*CURRENT)
000116                SIGNATURE('23102015 V1')
000117 /*********************************************************************/
000118 /*   *SRVPGM      B01U0001     NEBAR        23/10/15  16:11:58      */
000119 /*********************************************************************/
000120                 EXPORT     SYMBOL("PUSHLIB")
000121                 EXPORT     SYMBOL("POPLIB")
000123                 EXPORT     SYMBOL("CHECK_SQLSTATE")
000124                 EXPORT     SYMBOL("EXISTLIBRARY")
000125                 EXPORT     SYMBOL("ADDLIBLE")
000126                 EXPORT     SYMBOL("RTVLIBL")
000127                 EXPORT     SYMBOL("ISVALIDNUMERICDATE")
000128              ENDPGMEXP
```

Figura 4.2.2.3.2. Lenguaje de enlazado para el programa de servicio B01U0001, miembro fuente : B01U0001.BND.

Finalmente, una vez creado, es necesario instalar el programa de servicio en el directorio de enlace que se ha descrito en este apartado, cuyo nombre es ***SRVPBND***, para ello es necesario hacerlo a través del comando ***ADDBNDIRE***, de esta forma será alojado como una entrada y podrá ser consumido por cualquier programa cliente que no necesite **(Diedrich, y otros, 2016, pág. 79)**. La figura 4.2.2.3.3 muestra la entrada en el directorio de enlace.

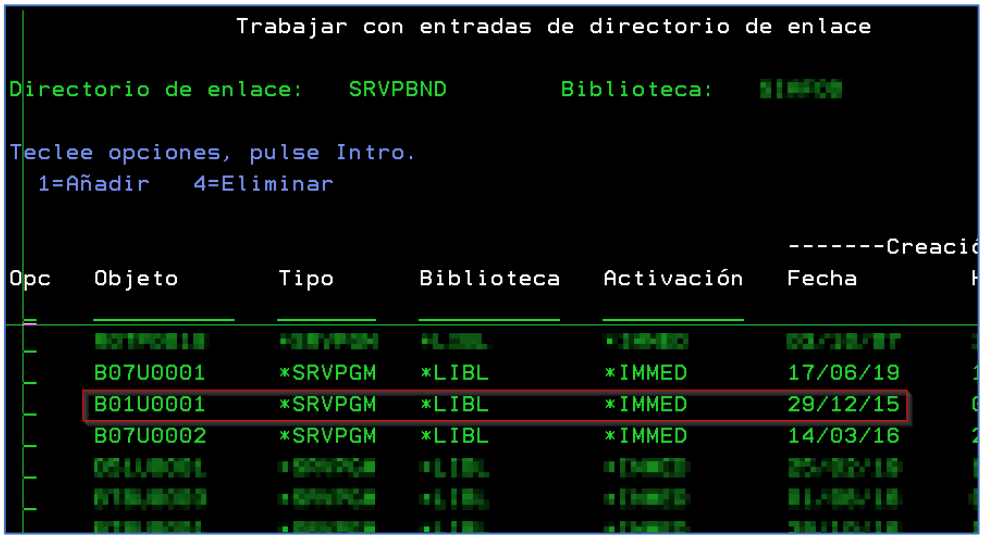


Figura 4.2.2.3.3. Programa de servicio B01U0001 instalado en el directorio de enlace SRVPBND.

A continuación, para mostrar la información del programa de servicio **B01U0001** dentro del sistema ,vamos a ejecutar el comando ***DSPSRVPGM*** para mostrar algunos de los detalles expuestos en este apartado, como son el objeto modulo, **M01U0001** objeto tipo ***MODULE,** del cual está compuesto y los subprocedimientos que ofrece como servicios, los cuales son exportados a través del lenguaje de enlazado que sirve para su creación al momento de ejecutar el comando ***CRTSRVPGM***. Las siguientes ilustraciones: Figuras 4.2.2.3.4, 4.2.2.3.5, muestran el resultado de la ejecución.

Figura 4.2.2.3.4. Resultado de aplicar el comando DSPSRVPGM al programa B07U0001, módulo que lo componen.

Figura 4.2.2.3.5. Resultado de aplicar el comando DSPSRVPGM al programa B01U0001, subprocedimientos exportados.

4.2.2.4 Acoplando la nueva arquitectura de software a la existente.

Uno de los aspectos más importantes y necesarios del desarrollo de software creado a partir de aplicaciones heredadas o también llamadas ***Legacy Applications***, es la integración con la arquitectura y el entorno ya existentes sobre el cual se va a instalar la nueva aplicación o modulo cuya arquitectura y diseño es diferente, en este caso el módulo o aplicación para el manejo de ***Check-List*** previo y durante el desembolso para una propuesta de crédito. La arquitectura existente con que se ha estado interactuando y cuyas grandes desventajas se han expuesto a la largo del presente trabajo, es una arquitectura monolítica, es ésta con la cual la mayoría de ***Legacy Applications*** o aplicaciones heredadas están construidas **(Amra, y otros, 2014, pág. 24)**, estas constan de

programas monolíticos que están compuestos de un solo modulo, en los cuales, se puede ver en un solo programa el código para el manejo de la interfaz de usuario, el acceso a la base de datos de la aplicación, la lógica de negocio, es decir, no hay un sentido de separación de responsabilidades **(Bedoya, Cruz, Lema, & Singkorapoom, 2016, p. 31)**, esto ya se ha expuesto en este capítulo cuando se analizan los programas de desembolso de un crédito del módulo SGCRED Sistema de Gestión de Créditos de una empresa financiera: **R7102310** y **R7101504**, los cuales están construidos bajo esta arquitectura y diseño, estos son programas monolíticos muy extensos sobre los cuales se ha tenido que acoplar o integrar la implementación del requerimiento de Check-List, cuyo diseño arquitectónico aplica los patrones arquitectónicos de diseño: **MVC** Modelo Vista Controlador y **SOA** Arquitectura Orientada a Servicios.

Uno de los nuevos componentes software con los que cuenta la nueva aplicación software para el requerimiento de Check-List son los programas de servicios, objetos tipo ***SRVPGM (Diedrich, y otros, 2016, pág. 78): B01U0001** y **B07U0002** los cual ya se han expuesto detenidamente para señalar la responsabilidad de cada uno de ellos, se analizó su diseño y estructura. Ahora, una vez construidos estos objetos, lo que resta es hacer uso de ellos y enlazarlos con los programas clientes que necesiten de sus servicios y con aquellos que ya existen el sistema principal de Empresa Financiera, el sistema SOFIA. Entonces, para completar toda la solución software para este requerimiento, es necesario la construcción de los programas clientes en los cuales se implementen la interacción del cliente con la aplicación, para esto es necesario una interfaz gráfica de usuario a través de la cual el usuario ingresa los datos que le son necesarios para el negocio y con las cuales la aplicación se alimenta. La

totalidad de los componentes mencionados de la aplicación Check-List se entrega a través de un formato de instalación o también llamado pase a producción al área de Sistemas-Producción de la Empresa Financiera, estos serán instalados en el sistema principal **SOFIA**, en la tabla 4.5 muestra los componentes software del pase a producción y en el Anexo B se detalla completamente el pase a producción, ya que en este se incluye los pasos a seguir por parte del operador para su correcta instalación.

Tabla 4.5

1. ARCHIVOS Y/O PROGRAMAS A PASAR						
Archivos			**Programas**			**Scripts**
Físicos/Lógicos	**Pantallas**	**Impresión**	**CL**	**RPGLE**	**SQLRGPLE**	**SQL**
NUEVOS						
	P71M0018			R01U0001 (COPY)	R71M0018	1. ALTERTABLE8051.sql
	P71C0018			R07U0001 (COPY)	R71C0018	2. UPDATEF8051.sql
	P71C0190	I71C0190			R71C0190	
	P71C0191	I71C0191			R71C0191	
					R71P0059	
					M01U0001 (MODULO)	
					M07U0002 (MODULO)	
MODIFICADOS						
F8051(sql)						

P7102310	R7102310
P7101504	R7101504
	R94C0001
P7102501	R7102501
RECOMPILADOS	
	R76C0054

Extracto del formato de pase a producción e instalación de la aplicación Check-List que se entrega al área de sistemas producción de Empresa Financiera.

En el extracto de formato de pase a producción que se expone en la tabla 5.4, se resaltan dos puntos importantes, en primer lugar, la categoría de los nuevos componentes y la de los modificados. La clasificación que se ha hecho, es importante por qué, los nuevos componentes representan la nueva aplicación Check-List cuyo diseño arquitectónico está basado en el que se está proponiendo en este trabajo, es decir, aplicar las nuevas técnicas de ingeniería de software y aprovechar patrones arquitectónicos; especialmente aquellos basados en capas cuyo principal objetivo es maximizar la reutilización de código y minimizar el impacto de cambios y mantenimiento **(Amra, y otros, 2014, pág. 34)**, adaptados a un paradigma de programación estructurado, tratando de evitar las características de mal diseño que posee aquella ya existente **(Amra, y otros, 2014, pág. 30)**, es decir, la arquitectura y diseño sobre la cual están construidos los componentes de la sección MODIFICADOS del formato de pase a producción de la tabla 5.4, arquitectura monolítica. **(Amra, y otros, 2014, pág. 24).**

A continuación, se va a exponer algunos de los componentes más importantes de la sección NUEVOS del formato de pase a producción, tabla 5.4, con la

finalidad de explicar la integración con la arquitectura existente y cómo hacer uso de la nueva para que trabajen de forma conjunta sin la necesidad de hacer cambios drásticos en el sistema SOFIA, ya que esto sería un proyecto de gran envergadura cuyo análisis queda fuera del presente trabajo.

- **Programa R71M0018:** Este programa, objeto de tipo ***PGM (Amra, y otros, 2014, pág. 79)**, cuya arquitectura es el modelo vista controlador **(Bass, Clements, & Kazman, 2013, p. 212)**, y orientada a servicios **(Bass, Clements, & Kazman, 2013, p. 222),** interactúa con el programa **R7102310** (Programa de desembolsos de créditos), objeto *PGM , el cual ha sido expuesto en la sección 4.2.1, desde el cual es invocado. Esta interacción o integración se realiza a través de un enlace dinámico del programa llamador, en este caso el programa **R7102310** con el programa **R71M0018**. Este proceso de enlace o ***binding***, es el proceso de creación de un programa, modulo o programa de servicio **(Diedrich, y otros, 2016, págs. 78-81)**, conceptos ya expuestos.

El enlace dinámico, quiere decir que el programa objetivo, el programa **R71M0018**, es descubierto e invocado en el momento de ejecución del programa **R7102310**, no hay una validación durante el proceso de compilación o ***binding*** que permita asegurar que este objeto ***PGM** exista, lo cual es una desventaja de usar el paradigma anterior, sin embargo, es necesario hacer ello por motivos de compatibilidad. La figura 4.2.2.4.1 muestra el diagrama de interacción del programa **R71M0018** con el programa **R7102310** y la figura 4.2.2.4.2 muestra el extracto del código fuente desde el cual es invocado.

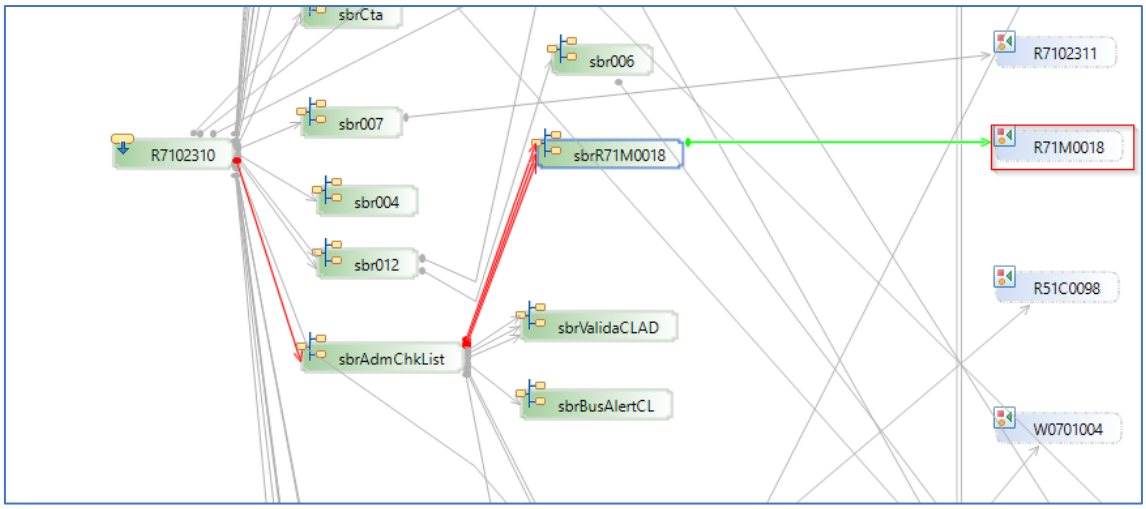


*Figura 4.2.2.4.1.*Diagrama de interacción del programa R71M0018 con el programa R7102310 de arquitectura monolítica.

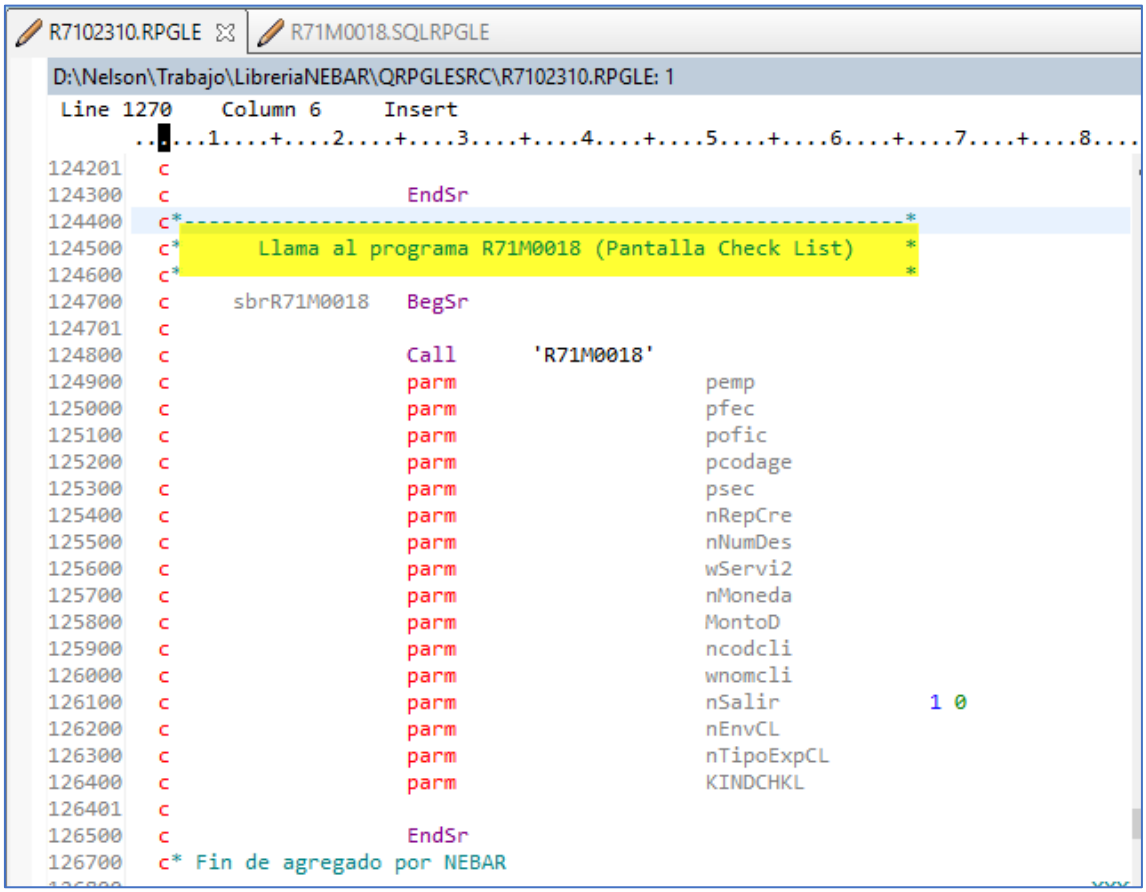


Figura 4.2.2.4.2. Invocación del programa R71M0018 a través del lenguaje RPG desde el programa R7102310.

Ahora se va a describir el aspecto funcional del programa y la forma cómo resuelve la necesidad del negocio, en este caso validad un conjunto de preguntas; condiciones necesarias para poder desembolsar un crédito, **Check-List** antes y durante este proceso de negocio. El programa **R71M0018** implementa todas estas reglas de negocio empleando para ello la arquitectura propuesta, toma un enfoque diferente para lograr su objetivo usando la misma tecnología, pero aprovechando su potencial en la medida de lo posible.

Se va a empezar exponiendo el resultado final de la solución ya resuelta en tiempo de ejecución, también se puede apreciar esto en el Anexo C; formato de pruebas unitarias enviado al área de sistemas de calidad. Como se ha

mencionado, este programa va a mostrar al usuario todas las preguntas necesarias a ser respondidas por el operador antes de realizar el desembolso, todas estas reglas de negocio deben de ser soportadas por este programa, este va a responder a ellas ya sea con respuesta positiva o negativa y de acuerdo con esto tomar una decisión, todo el detalle a este respecto se encuentra en el anexo a, descripción completa del requerimiento. Las figuras 4.2.2.4.3 y 4.2.2.4.4 muestran en tiempo de ejecución el programa **R71M0018** con la lista de preguntas para el **Check-List** previo al desembolso del crédito, estás deberán ser verificadas por el operador y deberá registrar un comentario respecto a lo acontecido finalizado el cuestionario.

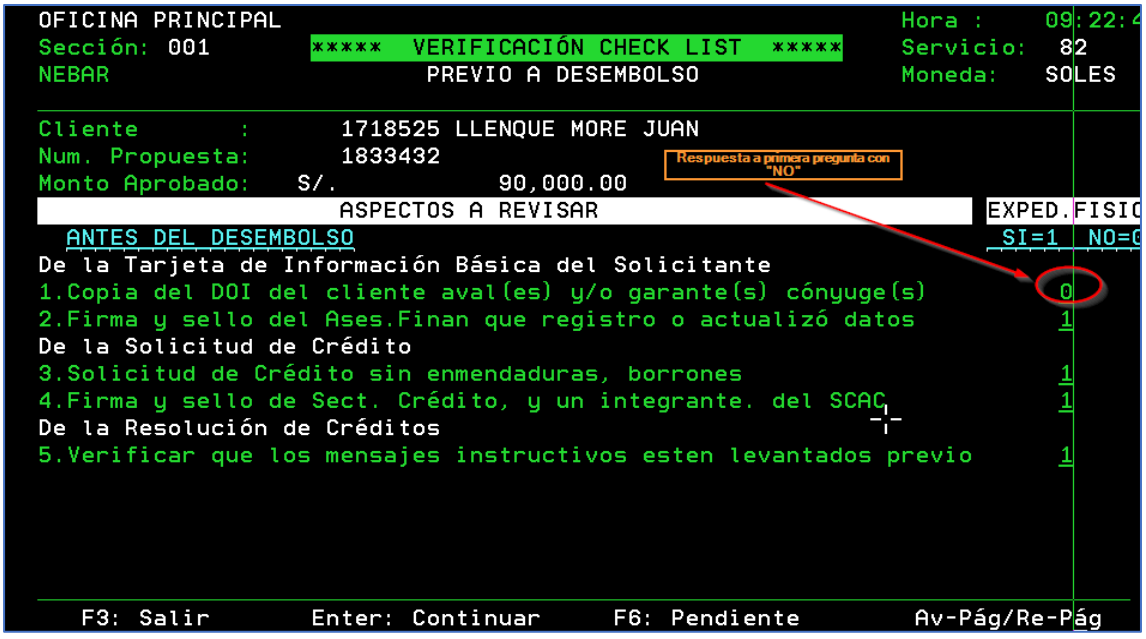


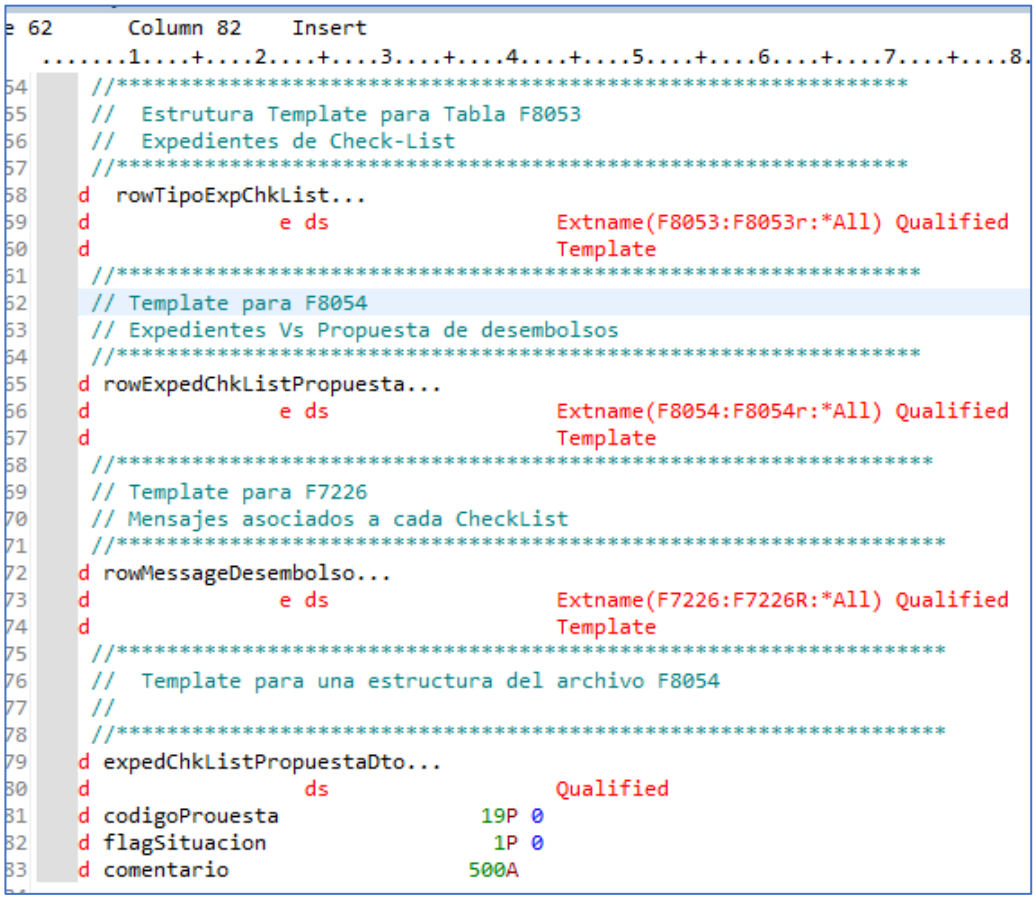
Figura 4.2.2.4.3. Lista de preguntas para el Check-List previo al desembolso de un crédito.

Figura 4.2.2.4.4. Check-List previo al desembolso del crédito, registrar estado final de respuestas.

Para el diseño del programa **R71M0018** se ha usado el patrón arquitectónico Modelo Vista Controlador **(Bass, Clements, & Kazman, 2013, p. 213)** y SOA, Arquitectura orientada a servicios **(Bass, Clements, & Kazman, 2013, p. 222)**. Para adaptar el patrón arquitectónico Modelo Vista Controlador en sus tres capas: el Modelo, la vista y Controlador, se ha hecho uso de los componentes expuestos previamente, los programas de servicio **B01U0001** y **B07U0002**, así como el uso del tipo de datos del lenguaje RPG llamado estructuras de datos **(Meyers & Buck, 2010, p. 91)**, las cuales permiten combinar múltiples tipos de datos en uno solo para lograr uno diferente, lo cual es de gran ayuda, ya que permite modelar las entidades del negocio, emular de alguna forma el paradigma de la programación orientada a objetos y hacer una mejor abstracción del diseño total de la aplicación, al combinar todos estos elementos se obtienen los

componentes que dicta el patrón arquitectónico **MVC**, Modelo Vista Controlador, los cuales están materializados a través de los siguientes componentes de los que consta el programa **R71M0018**:

- **Modelo**: El modelo representa toda la data de la aplicación, el estado de esta **(Bass, Clements, & Kazman, 2013, p. 212)**, la lógica del negocio y las reglas del negocio. Este componente está materializado a través de los programas de servicio **B01U0001** y **B07U0002**, el primero de ellos expone todos los servicios principales para la aplicación **Check-List**, y el segundo un conjunto de utilidades para consumir servicios del sistema operativo. Además, las entidades de negocio, algo así como los objetos en un paradigma de programación orientada a objetos, es representada a través de estructuras de datos las cuales que se encuentran en un archivo de código fuente llamado **R07U0002.RPGLE**, en la figura 4.2.2.4.5 se expone una porción de este con finalidad resaltar los puntos más importantes para demostrar su uso dentro del Controlador y del Modelo.



```
e 62      Column 82    Insert
  .......1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
54    //*****************************************************************
55    //  Estrutura Template para Tabla F8053
56    //  Expedientes de Check-List
57    //*****************************************************************
58   d  rowTipoExpChkList...
59   d                 e ds                  Extname(F8053:F8053r:*All) Qualified
60   d                                       Template
61    //*****************************************************************
62    // Template para F8054
63    // Expedientes Vs Propuesta de desembolsos
64    //*****************************************************************
65   d rowExpedChkListPropuesta...
66   d                 e ds                  Extname(F8054:F8054r:*All) Qualified
67   d                                       Template
68    //*****************************************************************
69    // Template para F7226
70    // Mensajes asociados a cada CheckList
71    //*****************************************************************
72   d rowMessageDesembolso...
73   d                 e ds                  Extname(F7226:F7226R:*All) Qualified
74   d                                       Template
75    //*****************************************************************
76    //  Template para una estructura del archivo F8054
77    //
78    //*****************************************************************
79   d expedChkListPropuestaDto...
80   d                  ds                   Qualified
81   d codigoProuesta                19P 0
82   d flagSituacion                  1P 0
83   d comentario                   500A
```

Figura 4.2.2.4.5. Estructuras para los layouts de los registros de tabla de la aplicación Check-List.

Vamos a explicar el detalle de las estructuras de que se muestran en la figura 4.2.2.4.5, en primer lugar, estas representan las entidades del negocio de aplicación Check-List, y hacen referencia para su diseño al diseño de la tabla que indican a través de la palabra clave del lenguaje **RPG *Extname* (Meyers & Buck, 2010, p. 96)**, la cual se puede apreciar en su declaración, es decir, que cada campo que compone la tabla será parte de la estructura en su diseño, sin embargo, estas no pueden ser usadas directamente, sino que sirven como plantilla para a

partir de ellas crear estructuras concretas, esto se logra a través de la palabra clave ***Template***, **(Diedrich, y otros, 2016, pág. 35)** de esta manera podemos emular de alguna forma el concepto de clase de la programación orientada a objetos, la cuales sirven como plantillas para crear objetos a partir de estas. Estas estructuras pueden ser usadas en cualquier programa que las necesite, a través de la inclusión del archivo de código fuente que las contiene. En los módulos que se han expuesto para la aplicación **Check-List**, **M07U0002** y **M01U0001** se hace uso de estas, esto se lograr a través del uso de la palabra clave **COPY** que permite referenciar a este archivo, como ya se ha expuesto, este módulo forma parte del programa de servicio **B07U0002** y cada procedimiento que se implementó se convirtió en un servicio. Las figuras 4.2.2.4.6, 4.2.2.4.7 y 4.2.2.4.8 muestran el extracto del código que importa el archivo que contiene las estructuras y la forma en la que cada procedimiento usa las usa cada una respectivamente.

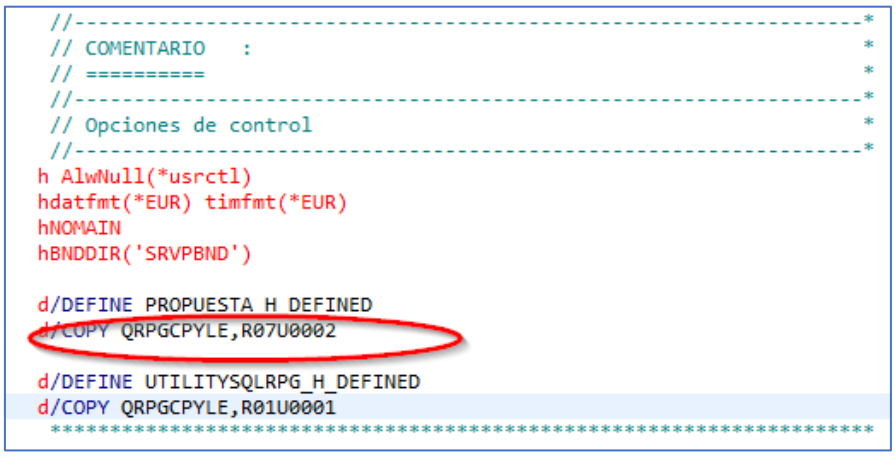
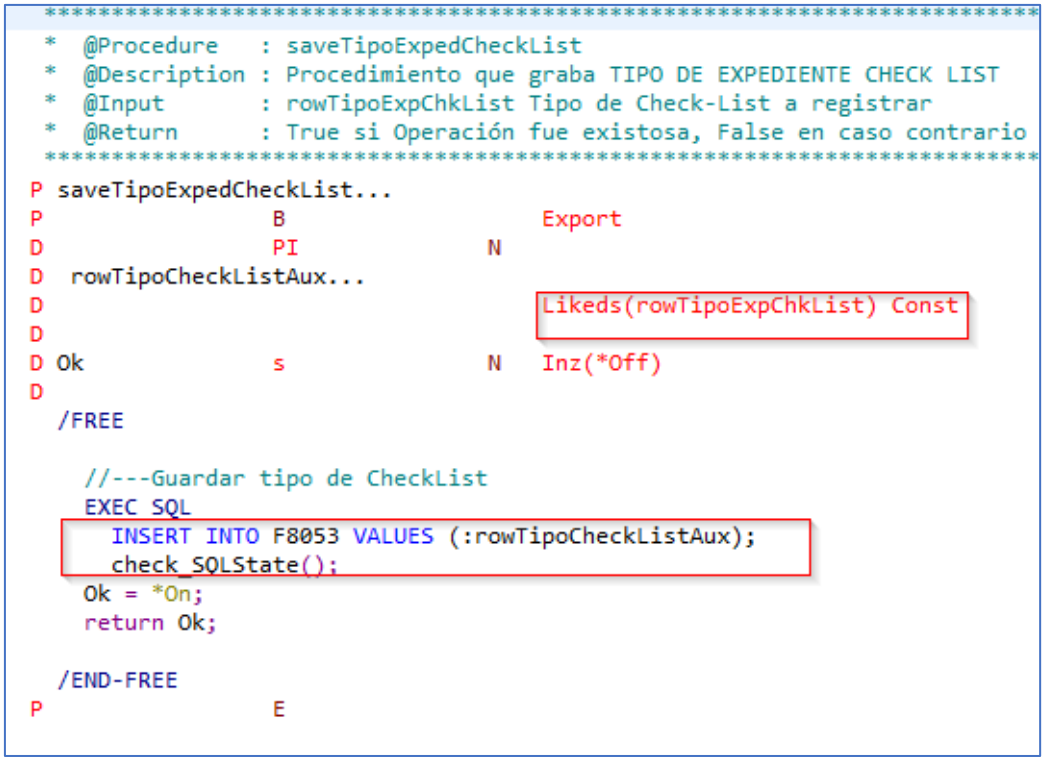


Figura 4.2.2.4.6. Inclusión de archivo fuente R07U0002 a través de la directiva COPY en el módulo M07U0002.



```
**************************************************************************
*  @Procedure   : saveTipoExpedCheckList
*  @Description : Procedimiento que graba TIPO DE EXPEDIENTE CHECK LIST
*  @Input       : rowTipoExpChkList Tipo de Check-List a registrar
*  @Return      : True si Operación fue existosa, False en caso contrario
**************************************************************************
P saveTipoExpedCheckList...
P                 B                   Export
D                 PI              N
D  rowTipoCheckListAux...
D                                     Likeds(rowTipoExpChkList) Const
D
D Ok              s               N   Inz(*Off)
D
  /FREE

    //---Guardar tipo de CheckList
    EXEC SQL
      INSERT INTO F8053 VALUES (:rowTipoCheckListAux);
      check_SQLState();
    Ok = *On;
    return Ok;

  /END-FREE
P                 E
```

Figura 4.2.2.4.7. Procedimiento saveTipoExpedCheckList del módulo M07U0002 que utiliza estructuras de datos del archivo R07U0002.

```
 * @Procedure    : deleteTipoExpedCheckList
 * @Description : Elimina un tipo de Check-List
 * @Input       : rowTipoExpChkList Tipo de Check-List a registrar
 * @Return      : True si Operación fue existosa, False en caso contrario
 ******************************************************************************
p deleteTipoExpedCheckList...
p                 B                   Export
D                 PI             n
d  rowTipoCheckListAux...
d                                     Likeds(rowTipoExpChkList) Const
d
d  numeroPropuesta...
d                 s              10  0
d  tipoCheckList  s               1  0
d  Ok             s                n   Inz(*Off)
d
  /FREE
      // Borrar tipo de Check List
      Exec Sql
      DELETE FROM F8053 WHERE COEXPRO = :rowTipoCheckListAux.COEXPRO
      AND PROPNRO = :rowTipoCheckListAux.PROPNRO
      AND TIPEXP = :rowTipoCheckListAux.TIPEXP;
      check_SQLState();
      OK = *ON;

      Return Ok;

  /END-FREE
P                 E
```

Figura 4.2.2.4.8. Procedimiento deleteTipoExpedCheckList del módulo M07U0002 que utiliza estructuras de datos del archivo R07U0002.

La figura 4.2.2.4.8 demuestra la forma cómo se importa un archivo en el cual se han declarado las estructuras de datos de datos que representan las entidades del negocio, por otro lado, las figuras 4.2.2.4.7 y 4.2.2.4.8 muestran el código fuente de la implementación de los procedimientos del módulo **M07U0002 saveTipoExpedCheckList** y **deleteTipoExpedCheckList** que hacen uso de estas, pero vamos a explicar aquí cada detalla de la forma cómo lo hacen, en primer lugar, el procedimiento **saveTipoExpedCheckList** en la declaración de su interfaz a través de la palabra reservada ***PI***, indica que va a recibir como argumento una estructura de datos del tipo **rowTipoExpChkList**

a través de la palabra reservada ***LikeDs,*** que quiere decir como la estructura de datos que se indica en el paréntesis, la cual representa la entidad de negocio tipo de **Check-List** para ser registrada en la base de datos, aquí se expresa el principio de una sola responsabilidad, evitando el problema de las aplicaciones o programa monolíticos que es el problema que se está resolviendo.

El mismo modelo de trabajo se expresa en el procedimiento **deleteTipoExpedCheckList**, el cual en la declaración de su interfaz indica que va a recibir como argumento una estructura de datos de tipo **rowTipoExpChkList** la cual va a dar de baja en la base de datos de la aplicación **Check-List**.

Se han expuesto dos subprocedimientos o procedimientos los cuales se encargan de todo el trabajo relacionado con los cambios de estado en el modelo de la aplicación, esto con la finalidad de demostrar la capa Modelo del patrón arquitectónico Modelo Vista Controlador, todos los subprocedimientos y su implementación se encuentra en el Anexo D. Este modelo es consumido por la capa Controlador la cual se va a exponer más adelante.

- **Vista:** La capa vista en el patrón arquitectónico Modelo Vista Controlador permite a la aplicación renderizar el modelo, solicitar actualizaciones de este, le permite al Controlador seleccionar la vista, solicitar acciones a ser ejecutadas desde este. **(Bass, Clements, & Kazman, 2013, p. 214)**, es decir permite al usuario interactuar con la aplicación, esta vista puede ser una interfaz de usuario Web, otro programa, puede ser un programa por lotes, y para el caso que estamos

tratando, una pantalla modelo 5250, también llamada pantalla verde. De todas las vistas de la aplicación **Check-List,** la que se va a exponer es la del programa **R71M0018** el cual estamos tratando como parte de la integración del marco de trabajo. La vista del programa **R71M00018**, tiene por nombre **P71M0018**, como se puede notar en el nombre se ha cambiado la letra R por P, esto se hace como convención de nomenclatura. Esta pantalla va a presentar los tipos de **Check-List** previo y durante el desembolso al usuario operador, y es utilizada por los controladores que la necesiten, en este caso el programa que funge de controlador es el R71M0018. El diseño de la pantalla se muestran en las figuras 4.2.2.4.9 y 4.2.2.4.10 y la forma como luce en tiempo de ejecución en las figuras 4.2.2.4.11 y 4.2.2.4.12 respectivamente.

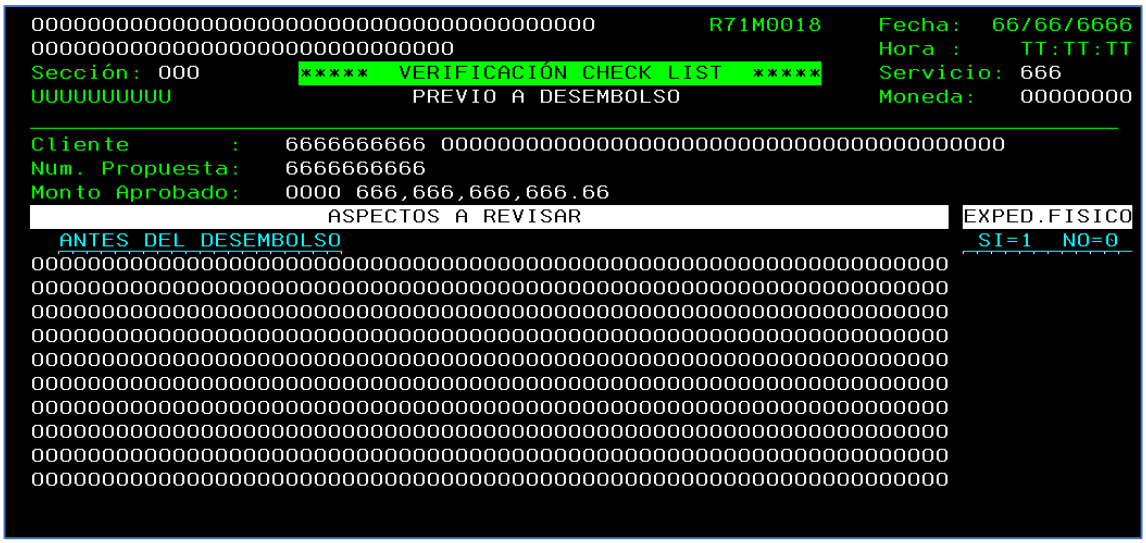


Figura 4.2.2.4.9. Diseño de pantalla P71M0018, Check-List Previo al desembolso.

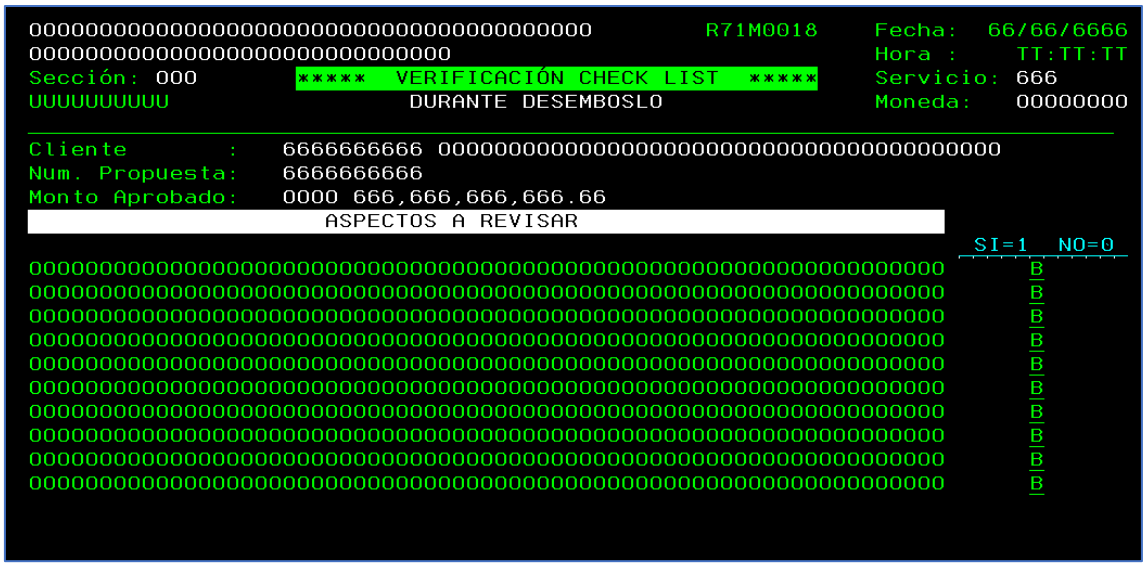


Figura 4.2.2.4.10. Diseño pantalla P71M0018, Check-List Durante el desembolso de un crédito.

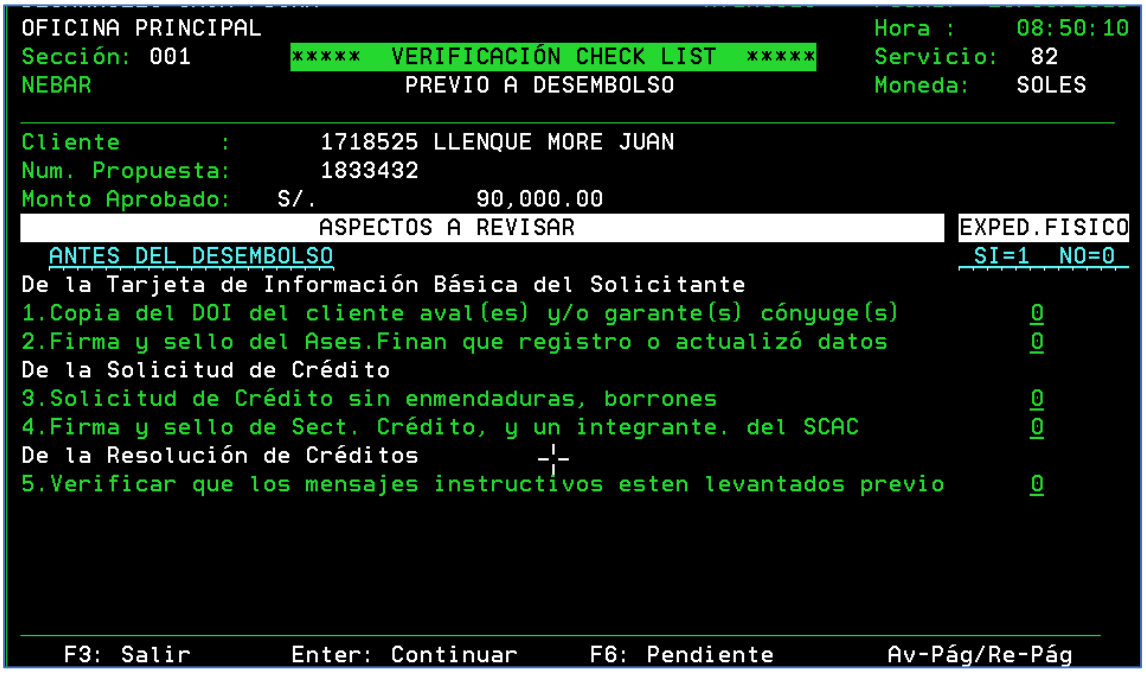


Figura 4.2.2.4.11. Pantalla P71M0018 en tiempo de ejecución, Check-List previo al desembolso.

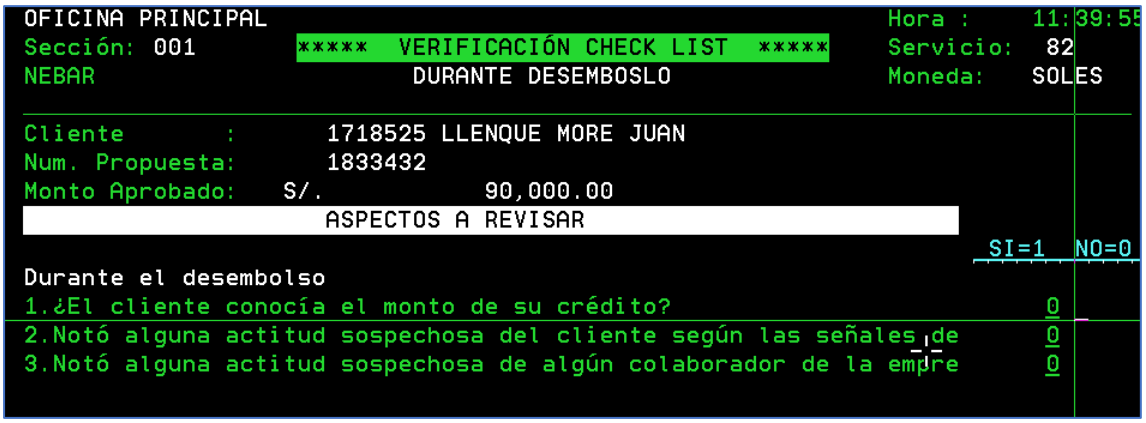


Figura 4.2.2.4.12. Pantalla P710018 en tiempo de ejecución, Check-List Durante el desembolso de un crédito.

Como se puede notar, ambos de tipos de Check-List son mostrados en el mismo archivo de pantalla, es decir, el archivo que contiene la vista, ya que ambos constan de un conjunto de preguntas y sólo difieren en la forma cómo se les denomina, es decir, previo y durante el desembolso. Aquí queda expresado la capa Vista del patrón arquitectónico Modelo Vista Controlador, el flujo total de todas las pantallas de la aplicación Check-List se muestran en el Anexo C, el cual contiene las pruebas unitarias y el flujo de estas a lo largo de la interacción con el operador al momento que este intenta realizar un desembolso de crédito a un cliente de la Empresa Financiera.

- **Controlador:** De acuerdo con Len Bass y otros autores, la capa Controlador del patrón arquitectónico Modelo Vista Controlador, se encarga de definir el comportamiento de la aplicación, mapear las acciones del usuario para actualizar el modelo y seleccionar la vista de

acuerdo a cada acción que realice el usuario como respuesta a estas **(Bass, Clements, & Kazman, 2013, p. 214)**. Uno de los controladores de la aplicación **Check-List** que se va a exponer como parte de la aplicación del nuevo Marco del Trabajo para la construcción de aplicaciones empresariales usando el lenguaje de programación **RPG**, es el programa **R71M0018**, objeto de tipo ***PGM (Diedrich, y otros, 2016, pág. 79)**, el cual se está tratando en este apartado. Este programa funge de controlador para la vista **P71M0018**, se encarga de seleccionar la vista del tipo de Check-List, ya sea el Check-List previo al desembolso de un crédito o durante este, que se debe de mostrar. También actualiza el modelo haciendo uso del programa de servicio **B07U0002**; cuya estructura ha sido expuesta en este trabajo, consumiendo para ello los servicios que este ofrece para lograr su tarea, además se encarga de crear las estructuras de datos concretas que modelan las entidades de negocio que se expusieron en el apartado de la capa Modelo, las cuales se encuentran declaradas en el archivo **R07U0001.RPGLE**. Es muy importante señalar que este componente es una pieza clave en todo el desarrollo del nuevo Marco de Trabajo, ya que este se encarga de integrar todas las partes que se han expuesto hasta este punto, y es un el lugar dónde se manejan todas las acciones del usuario con la aplicación y permite mostrar la forma cómo reflejar dichas acciones sobre el modelo de forma totalmente distinta a como se hacía usando el paradigma anterior de desarrollo de aplicaciones, es decir, construyendo programas monolíticos donde todas las reglas del negocio, el modelo y la vista se mezclaban sin tener un sentido de la

separación de responsabilidades, lo que conllevaba a un esfuerzo mucho mayor para su mantenimiento y la difícil tarea de escalar la aplicación rápidamente para atender nuevas solicitudes del negocio.

El controlador, programa **R71M0018**, consume los servicios del programa de servicio **B07U0002**, objeto de tipo ***SRVPGM (Bedoya, Cruz, Lema, & Singkorapoom, 2016, p. 78)**, referenciando en su directiva de compilación el directorio de enlace donde está instalado, objeto de tipo ***BNDIR (Bedoya, Cruz, Lema, & Singkorapoom, 2016, p. 79)**, estos servicios le permiten interactuar con el modelo para reflejar todas las acciones que el usuario realiza sobre este y mostrar dicho modelo a través de la vista **P17M0018**. La figura 4.2.2.4.13, en el punto número uno muestra el código fuente para referenciar el directorio de enlace cuyo nombre es **SRVPBND** a través de la palabra reservada **BNDIR**, ya expuesto con detalle en este trabajo, en el punto número dos se declara la referencia a la pantalla **P71M0018** y por último en el punto número tres el archivo ***copy*** **R07U0002** que contienen las estructuras de datos que sirven como plantilla para crear estructuras de datos concretas.

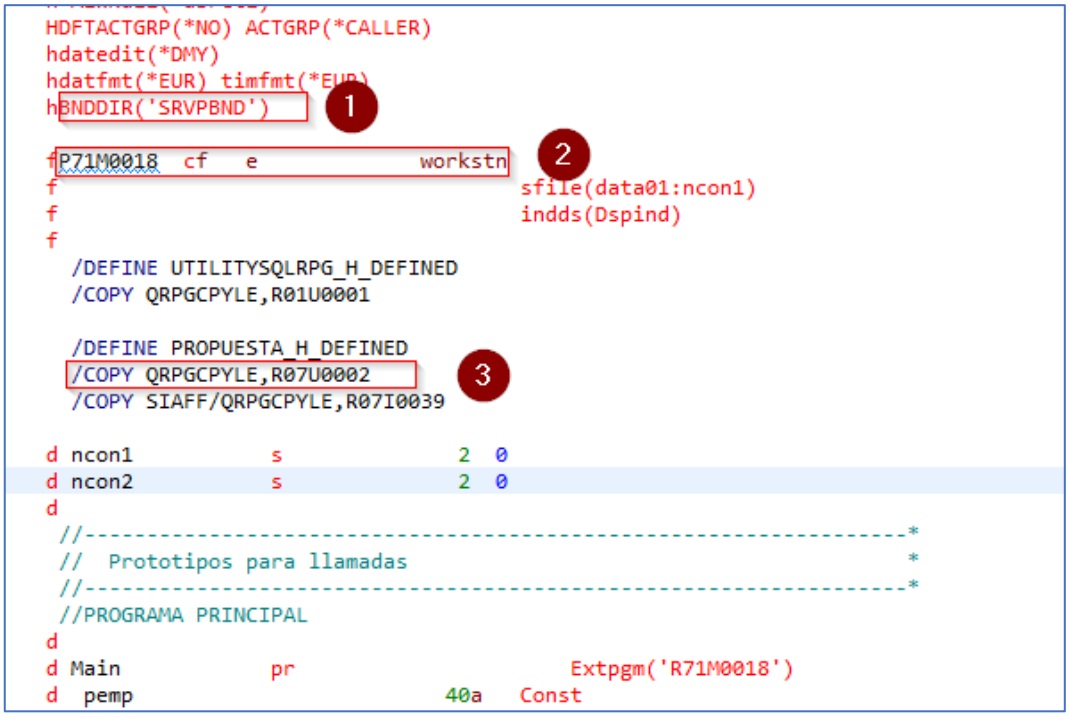


Figura 4.2.2.4.13. Controlador, Programa R71M0018, integración con la vista y el modelo de la aplicación Check-List.

Las estructuras de datos concretas usan como modelo para su construcción aquellas que sirven como plantilla y que se encuentran declaradas en el archivo **R07U0002**, esta acción de crear las estructuras de datos concretas podría compararse a la creación de objetos de una clase en el paradigma de la programación orientada a objetos. Luego son usadas como argumentos en los servicios que consume el controlador **R71M0018** del programa de servicio **BU7U0002** para interactuar con el modelo de la aplicación. La figura 4.2.2.4.14 muestra la construcción de las estructuras de datos en el programa **R71M0018**, las cuales referencian a sus respectivas plantillas usando la palabra

clave del lenguaje **RPG *likeDs*** que significa literalmente, como la estructura de datos.

```
 *  Program's Data Structures                        *
 ***************************************************
d  arrayTipoChkList...
d                 ds                  likeds(ChekList_Data_t) dim(99)
d dataTipChkList  ds                  likeds(ChekList_Data_t) Inz
d dsExpChkListPro...
d                 ds                  likeds(rowExpedChkListPropuesta) Inz
d expChkListDto   ds                  likeds(expedChkListPropuestaDto) Inz
d dsTipoExpChkList...
d                 ds                  likeds(rowTipoExpChkList) Inz
d dsMessaDesembolso...
d                 ds                  likeds(rowMessageDesembolso) Inz
d dsClienteDto    ds                  likeds(cliente_t) Inz
d
 ***************************************************
```

Figura 4.2.2.4.14. Estructuras de datos concretas creadas a partir de plantillas del archivo R07U008.RPGLE.

Los servicios que consume en controlador **R71M0018** y los cuales usan las estructuras de datos antes expuestas son los siguientes :

1. **Servicio getDesLarFromDigide**

   El servicio **getDesLarFromDigide** encapsula el acceso a los datos de la tabla de configuraciones de la aplicación Check-List, su implementación se encuentra en el módulo **M07U0002**, el cual forma parte del programa de servicio **B07U0002** y es a través de este último componente que llega a ser un servicio, ya que es quien se encarga de exportarlo, o hacerlo público para que pueda ser consumido declarando lo en el archivo de lenguaje de

enlazado expuesto **B07U0002.BND**. La implementación se muestra a continuación

```
p getDesLarFromDigide...
p                 B                   Export
d                 PI            80A   varying
d table                         10A
d digide                         3P 0
d codtab                         6A
d
d queryString     s             500A
d descripcionLarga...
d                 s             80A   Varying Inz(*blanks)
d
  /FREE

     queryString = 'SELECT DESLAP FROM '
             + %trim(table) + ' WHERE DIGIDE = '
             + %char(digide) + ' AND CODTAB = '+ QUOTE + codtab + QUOTE;

     Exec SQL PREPARE MySQLStatement FROM: queryString;

     Exec SQL Declare descripLarCS Cursor WITH HOLD for
                 MySQLStatement;

     Exec SQL Open descripLarCS;

     dou SQLSTATE = '02000';
         Exec SQL
            Fetch next from descripLarCS into: descripcionLarga;
         If SQLState = '02000' or SQLCode < *zeros;
          Leave;
         endif;
     Enddo;

     return descripcionLarga;

  /END-FREE
  p                 E
```

En la implementación del servicio, su interfaz declara que recibe tres argumentos, el primero es el nombre de la tabla de configuraciones, de nombre ***table***, cuyo tipo de datos es cadena de caracteres de longitud máxima de 10, y el segundo es código de configuración en la tabla de parámetros, ***digide***, cuyo tipo de dato es un valor numérico con capacidad para almacenar 3 dígitos, el tercer argumento represente el código de

parámetro a acceder, ***codtab***, cuyo tipo de datos es una cadena de caracteres de longitud máxima de 6. Retorna como valor la descripción de este parámetro ***descripcionLarga***, cuyo tipo de dato es una cadena de caracteres de longitud máxima de 80. Como se mencionó, RPGLE no necesita de una conexión de base de datos, ya que, al ser la base de datos DB2 parte del sistema operativo, no la necesita y ofrece una gran flexibilidad para trabajar directamente con las tablas de la base de datos a las cuales se desea acceder simplemente haciendo referencia a ellos en los programas donde se les necesite.

EL controlador **R71M0018** accede a este servicio llamándolo explícitamente como si fuese el método interno de una clase, tal como se hace en el paradigma de la programación orientada a objetos, esto es posible debido a que como ya se ha explicado, el programa de servicio **M07U0002** se encuentra instalado en el directorio de enlace de nombre ***SRVPBND***. El siguiente extracto de código fuente del programa controlador **R71M0018** muestra el uso de este servicio, por lo tanto, no se accede al modelo directamente, lo cual ya representa la diferencia entre el paradigma monolítico y el paradigma estructurado.

```
//-------------------------------------------------------------*
//  RUTINA PRINCIPAL                                           *
//-------------------------------------------------------------*
  tablaDigide=PARAM_TABLE;
  digide= LOAN_SCHEMA;
  codTabla=%EDITC(pServi1:'X');
  schema = getDesLarFromDigide(tablaDigide:digide:codTabla);
```

2. Servicio updateMessageDesembolso

El servicio **updateMessageDesembolso** encapsula la actualización de los mensajes que se generan antes y durante el desembolso de un crédito como

consecuencia de la respuesta a las preguntas del Check-List. Este se encuentra implementado en el módulo **M07U0002**, el cual forma parte del programa de servicio **B07U0002** y es a través de este componente que se convierte en un servicio, mediante el procedimiento que se ha explicado para el servicio **getDesLarFromDigide**. El código fuente de la implementación de este servicio se muestra a continuación.

```
 ****************************************************************
 * updateMessageDesembolso () : Actualiza archivo F7226         *
 * messageMessageToUpd = (Input) Mensaje de CheckList a Actualizar  *
 * updOk = (Output) Flag, True si todo fue Correcto, False      *
 *         caso contrario                                       *
 ****************************************************************
p updateMessageDesembolso...
p                 B                   Export
d                 PI              N
d messageMessageToUpd...
d                                     likeDs(rowMessageDesembolso) const
d updOk           s               N   Inz(*Off)
d
  /FREE
      Exec Sql
        update F7226 set USULEV =:messageMessageToUpd.USULEV,
          DIALEV =:messageMessageToUpd.DIALEV,
          MESLEV =:messageMessageToUpd.MESLEV,
          AÑOLEV =:messageMessageToUpd.AÑOLEV,
          OBSERV =:messageMessageToUpd.OBSERV
          where REPCRE =:messageMessageToUpd.REPCRE
          AND NUMDES =:messageMessageToUpd.NUMDES AND
          MENSAJ LIKE 'CHECK LIST DURANTE DESEMB.%';
          check_SQLState (); //Make sure that all is well
      updOk = *on;
      return updOk;

  /END-FREE
p                 E
```

La interfaz del servicio declara que recibe como argumento el mensaje de Check-List que va a actualizar, cuyo nombre ***messageMessageToUpd***, este argumento es de tipo de la estructura de datos ***rowMessageDesembolso***, el cual como se explicó es una de las plantillas para crear estructuras de datos concretas, y retorna un valor booleano luego de terminadas las operaciones que este hace, verdadero si no hubo problemas, falso en contrario, es

notorio que siempre se use al final de cada una operación ejecutada contra la base de datos, en este caso DB2, se use el procedimiento ***check_SQLState ()***, más adelante se va explicar este servicio. En el siguiente fragmento de código se muestra la forma cómo usa el controlador **R71M0018** este servicio, se nota que no se accede al modelo directamente dentro del mismo programa, lo cual era lo que se hacía con el paradigma de desarrollo de aplicaciones anteriores, lo que se hace es trasladar todos los valores de la vista a través de este controlador para luego llamar al servicio el cual realiza las tareas mencionadas.

```
zFecRem=%Timestamp(sFecha);
if nTipoExpCL=CHK_DURANTE_DESEMBOLOSO And   pFlgPresen=NO_PRESENCIAL;
       dsMessaDesembolso.REPCRE = pRepCre;
       dsMessaDesembolso.NUMDES = pNumDes;
       dsMessaDesembolso.USULEV = systemUser;
       dsMessaDesembolso.DIALEV = nDiaCal;
       dsMessaDesembolso.MESLEV = nMesCal;
       dsMessaDesembolso.AÑOLEV = nAnioCal;
       dsMessaDesembolso.OBSERV = %trim(WANTCOM);

       updateMessageDesembolso(dsMessaDesembolso);

       exsr sbrfin;
 endif;
```

3. Servicio getClienteDto

El servicio **getClienteDto** encapsula el acceso de los datos principales de un cliente de la empresa financiera, las tres últimas letras indican es para indicar que emula el patrón de diseño ***Data Transfer Object***, como señala Martin Fowler, esto reduce la cantidad del total de datos de un recurso a solo los necesarios, de esta forma se crea una estructura más pequeña que contenga solamente la información necesaria ya que no es necesario todo el registro de la tabla a la cual se está accediendo, sino solo aquellos datos necesarios para el trabajo que realiza quienes consuman este servicio

(Fowler, et al., 2011, p. 403). Este servicio se encuentra implementado dentro del módulo **M07U0002**, el cual forma parte del programa de servicio **B07U0002**, y al igual que los servicios ya expuestos este llega a convertirse a través del mismo procedimiento descrito para estos. En siguiente extracto de código fuente muestra la implementación del servicio.

```
*****************************************************************
* getClienteDto() : retorna los datos más importantes de un cliente *
*                   CMAC PIURA en una estructura de datos.          *
* codigoCli(input) : Código del cliente                             *
* clienteDto(output) : Estructura de datos con los datos del cliente*
*****************************************************************
p getClienteDto   B                   Export
d getClienteDto   PI                  likeDs(cliente_t)
d  codigoCli                    9P 0 Const
d
d  clienteDto     ds                  likeDs(cliente_t)
d
  /FREE
      Exec SQL
      SELECT TRIM(NOMBR1), TRIM(NOMBR2), TRIM(APEPAT),
      TRIM(APEMAT), TRIM(CLAPER) AS TIPO_PERSONA,
      CODCLI, AGEAPE, TRIM(NOMBCL),
      (CASE WHEN TIPDOC = '' THEN '7' ELSE TIPDOC END) AS TIP_DOI,
      (CASE WHEN CLAPER <> 'N' THEN TRIM(SIGLAS)
      ELSE DOCUME END) AS DOI into: clienteDto FROM F5101
      WHERE CODCLI = :codigoCli;
      check_SQLState ();

      return clienteDto;

  /END-FREE
p                 E
```

En la implementación del servicio, este declara en su interfaz, a través de la palabra reservada PI del lenguaje de programación RPG, que recibe como argumento el código de cliente a consultar ***codigoCli***, cuyo tipo de datos es valor numérico con capacidad para nueve dígitos, del cual se van a obtener sus datos o podríamos decir atributos como el termino usado en la programación orientada a objetos. Este retorna una estructura de datos de tipo ***cliente_t*** que agrupa los datos importantes del cliente. Para asegurarse que todo han ido correctamente usa el procedimiento ***check_SQLState()***. El

controlador **R71M0018** usa este servicio de la manera como lo muestra el siguiente extracto de código fuente de controlador, aquí se puede también apreciar que no se accede al modelo directamente, sino a través de la invocación del servicio.

```
If pmoned1 = SOLES;
 WMONEDA = SOLES_SYMBOL;
 WMONED1= LABEL_SOLES;
ElseIf pmoned1=DOLAR;
    WMONEDA = DOLAR_SYMBOL;
    WMONED1= LABEL_DOLAR;
//Other currencies
EndIf;
// Getting Customer's Data
dsClienteDto = getClienteDto(pCodCli);
ExSr sbrMostrarExpAntes;
```

4. Servicio getNumberMessagePropuesta

El servicio **getNumberMessagePropuesta** encapsula el acceso al número de mensajes de **Check-List** que contiene una propuesta de desembolso de crédito. Este servicio se encuentra implementado dentro del módulo **M07U0002** el cual forma es parte del programa de servicio **B07U0002**, como en los servicios anteriores, es a través de este componente que se convierte en un servicio aplicando el procedimiento ya descrito para los servicios anteriores. En el siguiente extracto de código se muestra la implementación del servicio.

```
 ****************************************************************
 * getNumberMessagePropuesta() : Obtiene el número de mensajes de *
 *                      Check-List que posee una propuesta        *
 *                      de desembolso de crédito.                 *
 * nroPropuesta = (input) número de propuesta                     *
 * nroDesembolso = (input) tipo de checklist                      *
 *  nroMessagePropuesta  = (output) Cantidad de mensajes          *
 *                       de Check-List de una propuesta           *
 ****************************************************************
P getNumberMessagePropuesta...
P                 B                   Export
 * Interfaz
D                 PI             5P 0
```

```
D  nroPropuesta                 10P 0 Const
D  nroDesembolso                 5P 0 Const
 *
D  nroMessagePropuesta...
D                 s              5P 0 INZ(0)
  /FREE

      Exec SQL
      SELECT COUNT(*) INTO : nroMessagePropuesta FROM F7226
      WHERE REPCRE= :nroPropuesta AND NUMDES= :nroDesembolso;
      check_SQLState();

      return nroMessagePropuesta;

  /END-FREE
P                 E
```

En la implementación del servicio, este declara a través de su interfaz, palabra reservada del lenguaje RPG ***PI***, dos argumentos, el primero de ellos es el número de propuesta, ***nroPropuesta***, cuyo tipo de dato es un valor numérico con capacidad para diez dígitos, el segundo argumento es el número de desembolso, ***nroDesembolso***, cuyo tipo de dato es un valor numérico con capacidad para almacenar 5 dígitos. El valor de retorno es el número de mensajes, ***nroMessagePropuesta***, cuyo tipo de dato es un valor número con capacidad para cinco valores. Ya que al igual que la mayoría de los servicios interactúa con la base de datos, este revisa que todo vaya correctamente usando el servicio ***check_SQLState()***, el cual se encuentra implementado en el programa de servicio **M01U0001**. El controlador **R71M0018** usa este servicio de la manera como lo muestra en siguiente extracto de código el cual le pertenece, aquí también se nota la separación de responsabilidades que debe de tener cada componente.

```
If ProcMsgBox('Pregunta':sMensaje:'05')='06';
   pEnvCL=1;
   ExSr sbrGrabarExp;
   If nTipoCL=CHK_PREVIO_DESEMBOLSO;
      nContMsgProp =  getNumberMessagePropuesta(pRepCre:pNumDes)
                    + 1;
```

5. **Servicio saveMessageDesembolso**

El servicio **saveMessageDesembolso** encapsula la persistencia en la base de datos de un mensaje de **Check-List** asociado a una propuesta de desembolso de crédito, sea éste previo o durante el mismo. Este servicio se encuentra implementado dentro del módulo **M07U0002**, este forma parte del programa de servicio **B07U0002**, y es mediante este último componente que llega a convertirse en un servicio a través del procedimiento explicado para con el resto de servicio de esta sección. El siguiente código fuente muestra la implementación del servicio.

```
 ****************************************************************
 * saveMessageDesembolso() : Guarda un registro en Archivo F7226 *
 * dsMessageDesembolso(input) : estructura de datos a insertar   *
 * flagSaveOk(Output) : Flag, True si todo fue correcto, False   *
 *                      caso contrario                           *
 ****************************************************************
p saveMessageDesembolso...
p                 B                   Export
d                 PI             n
d dsMessageDesembolso...
d                                     likeDs(rowMessageDesembolso) const
d flagSaveOk      s               n   Inz(*off)
d
  /FREE

    Exec SQL
      INSERT INTO F7226 VALUES(:dsMessageDesembolso);
      flagSaveOk = *On;
      check_SQLState();
      Return flagSaveOk;

  /END-FREE
p                 E
```

El servicio **saveMessageDesembolso** declara en su interfaz, usando la palabra reservada del lenguaje **RPG PI**, un argumento, este es una estructura de datos, ***dsMessageDesembolso,*** de tipo ***rowMessageDesembolso***, la cual es una de las estructuras que sirven como

plantilla y que se encuentra en el archivo **R07U0002**, el cual es ya se ha explicado. Retorna un valor booleano, ***flagSaveOk***, para indicar el resultado de la operación, si esta fue exitosa retornara verdadero, falso en caso contrario. Para verificar que la operación ha ido bien invoca el servicio ***check_SQLState()***, el cual pertenece al programa de servicio **B01U0001**. El controlador **R71M0018** usa de la siguiente manera como lo muestra el siguiente extracto de su código fuente el servicio en cuestión. Se puede notar que el controlador no accede directamente al modelo tal como lo hacía la metodología anterior, sino que se usa toda la infraestructura creada por el nuevo marco de trabajo.

```
saveMessageDesembolso(dsMessaDesembolso);
  isSaveCHLPREV = *on;
  iSalir = true;
Else;
   //envia mensaje de alerta
   sMensaje='CHECK LIST PREVIO PROPUESTA:'+
        %char(pRepCre)+'. CLIENTE:'+%char(pCodCli);
  nFlgRes=2;
  dsMessaDesembolso.REPCRE = pRepCre;
  dsMessaDesembolso.NUMDES = pNumDes;
  dsMessaDesembolso.CORREL = nContMsgProp;
  dsMessaDesembolso.MENSAJ = sMensaje;
  dsMessaDesembolso.MONTOR = pMontoD;
  dsMessaDesembolso.MONEDA = pmoned1;
  dsMessaDesembolso.DIAREG = nDiaCal;
  dsMessaDesembolso.MESREG = nMesCal;
  dsMessaDesembolso.AÑOREG = nAnioCal;
  dsMessaDesembolso.FLGRES = nFlgRes;
  dsMessaDesembolso.USUREG = systemUser;

  saveMessageDesembolso(dsMessaDesembolso);
```

6. Servicio updateExpedCheckListPropuesta

El servicio **updateExpedCheckListPropuesta** encapsula la abstracción de la actualización de los datos del Check-List de una propuesta de desembolso de crédito. Este servicio arma una consulta dinámica en su

implementación con la finalidad de hacer más flexible su uso por parte del programa llamador. Este servicio se encuentra implementado en el módulo **M07U0002** el cual forma parte del programa de servicio **B07U0002**, ese a través de este componente que llega a convertirse un servicio que puede ser llamado por cualquier programa cliente que lo invoque mediante el procedimiento ya expuesto para los servicios expuestos en este apartado. El siguiente código fuente muestra su implementación.

```
 ****************************************************************
 * updateExpedCheckListPropuesta()                              *
 * rowExpedChkListProP(Input) : Estructura de datos de expediente    *
 *                              de Check-List                   *
 * successfully (Output) : Flag, True todo correcto, False      *
 *                         caso contrario                       *
 ****************************************************************
P updateExpedCheckListPropuesta...
P                 B                   Export
d                 PI              n
d rowExpedChkListProP...
d                                     likeDs(rowExpedChkListPropuesta)
 * Stand Alone Variables
d successfully    s                n   Inz(*Off)
d queryStringUpd  s              500a
d fieldsToChange  s              400a   Varying
  /Free

     queryStringUpd = 'UPDATE F8054 SET ';
     If (rowExpedChkListProP.NUMDES <> *Zeros);
         fieldsToChange = ' NUMDES = '
           + %char(rowExpedChkListProP.NUMDES);
     EndIf;
     If (rowExpedChkListProP.TIPCHK <> *Zeros);
         If (%len(fieldsToChange) > *Zeros);
           fieldsToChange = %trim(fieldsToChange) + ', TIPCHK = '
           + %char(rowExpedChkListProP.TIPCHK);
         Else;
           fieldsToChange = %trim(fieldsToChange) + ' TIPCHK = '
           + %char(rowExpedChkListProP.TIPCHK);
         EndIf;
     EndIf;
     If (rowExpedChkListProP.FLGSIT <> *Zeros);
         If (%len(fieldsToChange) > 0);
           fieldsToChange = %trim(fieldsToChange) + ', FLGSIT = '
           + %char(rowExpedChkListProP.FLGSIT);
         Else;
             fieldsToChange = %trim(fieldsToChange) + ' FLGSIT = '
           + %char(rowExpedChkListProP.FLGSIT);
         EndIf;
     EndIf;
     If (rowExpedChkListProP.FLGEST <> 0);
```

```
   If (%len(fieldsToChange) > 0);
      fieldsToChange = %trim(fieldsToChange) + ', FLGEST = '
      + %char(rowExpedChkListProP.FLGEST);
   Else ;
        fieldsToChange = %trim(fieldsToChange) + ' FLGEST = '
      + %char(rowExpedChkListProP.FLGEST);
   EndIf;
EndIf;
If (rowExpedChkListProP.COMREV <> *Blanks);
   If (%len(fieldsToChange) > 0);
      fieldsToChange = %trim(fieldsToChange) +', COMREV = '
      + QUOTE + rowExpedChkListProP.COMREV + QUOTE;
      Else;
        fieldsToChange =%trim(fieldsToChange) + ' COMREV = '
      + QUOTE + rowExpedChkListProP.COMREV + QUOTE;
    EndIf;
EndIf;
If (rowExpedChkListProP.USRREV <> *Blanks);
   If (%len(fieldsToChange) > 0);
     fieldsToChange = %trim(fieldsToChange) + ', USRREV = '
     + QUOTE + rowExpedChkListProP.USRREV + QUOTE;
   Else;
     fieldsToChange = %trim(fieldsToChange) + ' USRREV = '
     + QUOTE + rowExpedChkListProP.USRREV + QUOTE;
   EndIf;
EndIf;
If (%char(rowExpedChkListProP.FECREV) <> DEFAULT_TIMESTAMP);
   If (%len(fieldsToChange) > 0);
      fieldsToChange =%trim(fieldsToChange) + ', FECREV = '
      + QUOTE + %char(rowExpedChkListProP.FECREV) + QUOTE;
   Else;
      fieldsToChange = %trim(fieldsToChange)+ ' FECREV = '
      + QUOTE + %char(rowExpedChkListProP.FECREV) + QUOTE;
   EndIf;
EndIf;
If (rowExpedChkListProP.USRLEV <> *Blanks);
   If (%len(fieldsToChange) > 0);
      fieldsToChange = %trim(fieldsToChange) + ', USRLEV = '
      + QUOTE + rowExpedChkListProP.USRLEV + QUOTE;
   Else;
      fieldsToChange =%trim(fieldsToChange) + ' USRLEV = '
      + QUOTE + rowExpedChkListProP.USRLEV + QUOTE;
    Endif;
EndIf;
If (%char(rowExpedChkListProP.FECLEV) <> DEFAULT_TIMESTAMP);
   If (%len(fieldsToChange) > 0);
      fieldsToChange = %trim(fieldsToChange) + ', FECLEV = '
      + QUOTE + %char(rowExpedChkListProP.FECLEV) + QUOTE;
   Else;
        fieldsToChange = %trim(fieldsToChange) + ' FECLEV = '
      + QUOTE + %char(rowExpedChkListProP.FECLEV) + QUOTE;
    EndIf;
EndIf;
If (rowExpedChkListProP.USRMOD <> *Blanks);
   If (%len(fieldsToChange) > 0);
     fieldsToChange = %trim(fieldsToChange) + ', USRMOD = '
     + QUOTE + rowExpedChkListProP.USRMOD + QUOTE;
   Else;
     fieldsToChange = %trim(fieldsToChange) + ' USRMOD = '
```

```
            + rowExpedChkListProP.USRMOD;
         EndIf;
      EndIf;
      If (%char (rowExpedChkListProP.FECMOD) <> DEFAULT_TIMESTAMP);
         If (%len(fieldsToChange) > 0);
            fieldsToChange = %trim(fieldsToChange) + ', FECMOD = '
            + QUOTE + %char(rowExpedChkListProP.FECMOD) + QUOTE;
         Else;
              fieldsToChange = %trim(fieldsToChange) + ' FECMOD = '
            + QUOTE + %char(rowExpedChkListProP.FECMOD) + QUOTE;
         EndIf;
      EndIf;
      queryStringUpd = %trim(queryStringUpd) + ' '
                     + %trim(fieldsToChange) + ' WHERE '
                     + 'PROPNRO = ' + %char(rowExpedChkListProP.PROPNRO)
                     + ' AND TIPCHK = '
                     + %char(rowExpedChkListProP.TIPCHK);
     //Para ejecutar sentencia, name statement should be unique
     Exec SQL
         EXECUTE immediate :queryStringUpd;
      successfully = *On;
      Return successfully;

  /End-Free
P                 E
```

Este servicio, **updateExpedCheckListPropuesta**, declara en su interfaz que recibe un argumento, ***rowExpedChkListProP***, el cual es tipo de la estructura de datos ***rowExpedChkListPropuesta***, la cual representa una propuesta de desembolso con su tipo de expediente de Check-List, retorna un valor booleano, ***successfully***, que indica si todo ha ido correctamente con valor verdadero, y falso en caso contrario. Luego de ejecutar la operación de actualización verifica que todo ha ido correctamente a través de la invocación del servicio ***check_SQLState()***, el cual pertenece a otro programa de servicio, en este caso **M01U0001**, lo cual demuestra que un programa de servicio puede hacer uso de otros programas de servicio para llevar a cabo mejor su trabajo. Haciendo de esta forma el trabajo de operaciones con la base de datos, podemos separar todas las tareas relacionadas con ella y lograr una mejor abstracción y un mejor diseño al evitar mezclar las responsabilidades. El controlador **R71M00018** utiliza

este servicio de la manera como lo demuestra en siguiente extracto de su código fuente y también un ejemplo de cómo otros programas clientes pueden hacer uso de este.

```
clear dsExpChkListPro; //Limpiando estructura
// Asiganando Valores
dsExpChkListPro.FLGSIT = nFlgSit;
dsExpChkListPro.USRMOD = systemUser;
dsExpChkListPro.COMREV = %trim(WANTCOM);
dsExpChkListPro.FECMOD = zFecRem;
dsExpChkListPro.USRREV = systemUser;
dsExpChkListPro.FECREV = zFecRem;
dsExpChkListPro.PROPNRO = pRepCre;
dsExpChkListPro.TIPCHK = nTipoCL;

 updateExpedCheckListPropuesta(dsExpChkListPro);
```

7. Servicio deleteTipoExpedCheckList

El servicio **deleteTipoExpedCheckList** encapsula en su implementación la abstracción de la eliminación de un tipo de Check-List relacionado a una propuesta de desembolso de crédito. Este servicio se encuentra dentro del módulo **M07U0002**, el cual forma parte del programa de servicio **B07U0002** y es a través de este componente de la aplicación que se convierte en un servicio mediante el procedimiento ya descrito, pues es la misma dinámica para cada subprocedimiento de un módulo que así se desea se comporte. El siguiente código fuente muestra la muestra la implementación de este servicio.

```
 ******************************************************************
 * @Proceder    : deleteTipoExpedCheckList
 * @Description : Elimina un tipo de Check-List
 * @Input       : rowTipoExpChkList Tipo de Check-List a registrar
 * @Return      : True si Operación fue exitosa, False en caso
 *                contrario
 ******************************************************************
P deleteTipoExpedCheckList...
p                 B                   Export
D                 PI             n
d  rowTipoCheckListAux...
d                                     LikeDs(rowTipoExpChkList) Const
```

```
d
d  Ok              s                n   Inz(*Off)
d
  /FREE
      // Borrar tipo de Check-List
      Exec Sql
      DELETE FROM F8053 WHERE COEXPRO = :rowTipoCheckListAux.COEXPRO
      AND PROPNRO = :rowTipoCheckListAux.PROPNRO
      AND TIPEXP = :rowTipoCheckListAux.TIPEXP;
      check_SQLState();
      OK = *On;

      Return Ok;
  /END-FREE
P                 E
```

Este procedimiento declara en su interfaz, mediante la palabra reservada ***PI*** del lenguaje de programación RPG, que recibe como un argumento, el cual es una estructura de datos, ***rowTipoCheckListAux***, la cual es del tipo ***rowTipoExpChkList***, que representa el expediente de Check-List asociado a una propuesta de crédito, de esto se puede notar que, se puede emular de alguna manera la forma de abstracción del paradigma de la programación orientada a objetos y lograr una mejor abstracción y diseño para las funcionalidades de la aplicación. Podríamos pensar de esta manera de la estructura de datos como objeto que deseamos eliminar de nuestro modelo, esto es más claro y conciso según las nuevas técnicas y prácticas de la ingeniería de software. A continuación, se muestra un extracto de código fuente de la forma como el controlador **R71M0018** utiliza este servicio dentro de su propia implementación.

```
ver sbrGrabarExpParcial;

crear dsTipoExpChkList;
//Asignando Valores
dsTipoExpChkList.COEXPRO = nCodExp;
dsTipoExpChkList.PROPNRO = pRepCre;
dsTipoExpChkList.TIPEXP = nTipoCL;
deleteTipoExpedCheckList(dsTipoExpChkList);
```

8. Servicio saveTipoExpedCheckList

El servicio **saveTipoExpedCheckList** encapsula la implementación de la persistencia en la base de datos de un tipo de un tipo Check-List asociado a una propuesta de desembolso de crédito. Este servicio se encuentra implementado en el módulo **M07U0002**, el cual forma del programa de servicio **B07U0002**, y este último componente que permite a este subprocedimiento llegar a ser un servicio, este se logra mediante el procedimiento ya expuesto para los primeros servicios de esta sección. El siguiente código fuente muestra la implementación de este servicio.

```
*********************************************************************
*  @Procedure  : saveTipoExpedCheckList                             *
*  @Description: Procedimiento que graba TIPO DE EXPEDIENTE CHECK LIST*
*  @Input      : rowTipoExpChkList Tipo de Check-List a registrar.  *
*  @Return     : True si Operación fue existosa, False en otro caso.*
*********************************************************************
P saveTipoExpedCheckList...
P                 B                   Export
D                 PI             N
D  rowTipoCheckListAux...
D                                     Likeds(rowTipoExpChkList) Const
D
D Ok              s               N   Inz(*Off)
D
  /FREE

    //---Guardar tipo de CheckList
    EXEC SQL
      INSERT INTO F8053 VALUES (:rowTipoCheckListAux);
      check_SQLState();
    Ok = *On;
    return Ok;

  /END-FREE
P                 E
```

El procedimiento declara en su interfaz, mediante la palabra reservada del lenguaje RPG ***PI***, que recibe como argumento una estructura de datos, ***rowTipoCheckListAux***, cuyo tipo es ***rowTipoCheckListAux***, la cual

representa un tipo de Check-List asociado a una propuesta de crédito, esta contiene toda la información necesaria a persistir en la base de datos. Al final de realizar la operación verifica que todo haya ido correctamente invocando el servicio ***check_SQLState()***, del programa de servicio **B01U0001**. El controlador **R71M0018** consume este servicio de la manera como lo muestra el siguiente extracto de código fuente de dicho controlador.

```
If WCODDOC <> 0;

   Clear dsTipoExpChkList;
   dsTipoExpChkList.COEXPRO = nCodExp;
   dsTipoExpChkList.PROPNRO = pRepCre;
   dsTipoExpChkList.TIPEXP = nTipoCL;
   dsTipoExpChkList.CODDOC = WCODDOC;
   dsTipoExpChkList.RPTCHK = nRptaUser;
   dsTipoExpChkList.USRREG = systemUser;
   dsTipoExpChkList.FECREG = zFecRem;

   saveTipoExpedCheckList(dsTipoExpChkList);

EndIf;
```

9. Servicio getDataChecklist

El servicio **getDataChecklist** encapsula la implementación de obtención de datos de un expediente de Check-List asociado a una propuesta de desembolso de crédito. Recoge todas preguntas y respuestas de las cuales está compuesto de acuerdo con su tipo, previo y durante un desembolso. Como se explicó en el apartado relacionado al requerimiento que resuelve la aplicación de Check-List, cada uno consta de un conjunto de preguntas que hace el operador al usuario previo y durante el desembolso. Esta regla de negocio está implementada en este servicio, el cual forma parte del módulo **M07U0002** el cual a su vez forma parte del programa de servicio **B07U0002**, al formar parte de ese último componente le permite

convertirse en un servicio para ser consumido por cualquier programa cliente que lo necesite. El procedimiento usado para lograr esto es el mismo que se ha aplicado para los primeros servicios que se han explicado en esta sección. El código fuente de su implementación es el que se muestra a continuación.

```
 ****************************************************************
 * getDataChecklist: Retorna preguntas y respuestas de Check-List*
 * propuestaNumber = (Input) número de propuesta                 *
 * tipoChkList = (Input) tipo de checklist                       *
 *             = (Output) Vector de Datos                        *
 * tipoPersona(Input) = puede ser :                              *
 *                  'N':  NATURA                                 *
 *                  'J': JURÍDICA                                *
 ****************************************************************
P getDataChecklist...
P                 B                   Export
d                 PI                  likeDs(ChekList_Data_t) Dim(99)
d  propuestaNumber...
d                                10P 0 Const
d  tipoChkList                    1P 0 Const
d  tipoPersona                    1A   Const
d
d  ArrdataCheckList...
d                 ds                  likeDs(ChekList_Data_t) Dim(99)
d
d  dataCheckListAux...
d                 ds                  likeDs(ChekList_Data_t)
d
d  count          s                2P 0
d
  /Free
    //DIRECTIVAS SQL
    Exec SQL Set Option Commit=*None, Naming = *Sys;
    Exec SQL Set Path=*Libl;

    count = 0;

    Exec SQL
      Declare ChkList_Cursor CURSOR FOR
      SELECT T.*, C.DESLAP, C.VALENT FROM
      (SELECT A.DOCCHKLIST_TIPOCATEGORIA AS KIND_CATEGORY,
      Row_Number() over() AS INDICE, B.DOCCHKLIST_CODIGODOCCHKLIST
      ,A.DOCCHKLIST_DESCRIPCION, (CASE WHEN C.DETEXPPROP_RPTA
      IS NULL THEN 0 WHEN C.DETEXPPROP_RPTA = 4 THEN 0
      ELSE 1 END) AS ANSWER_SYSTEM, A.DOCCHKLIST_ACEPTARESPUESTANO
      FROM F8051 A INNER JOIN F8052 B
      ON A.DOCCHKLIST_CODIGODOCCHKLIST = B.DOCCHKLIST_CODIGODOCCHKLIST
      LEFT JOIN F8053 C ON
      B.DOCCHKLIST_CODIGODOCCHKLIST = C.DOCCHKLIST_CODIGODOCCHKLIST
```

```
      AND C.PROPNUMERO = :propuestaNumber
      WHERE B.CHKLIST_TIPOCHKLIST = :tipoChkList
      AND LOCATE(:tipoPersona,TRIM(A.DOCCHKLIST_PERSONA)) > 0
      ) T, F6209 C
      WHERE C.DIGIDE = 643 AND CODTAB = VARCHAR(T.KIND_CATEGORY)
      for fetch only;

   Exec Sql Open ChkList_Cursor;
   Exec Sql Fetch ChkList_Cursor Into: dataCheckListAux;

   Dow SqlCode = 0;
    count += 1;
    ArrdataCheckList(count).kindCategory=dataCheckListAux.kindCategory;
    ArrdataCheckList(count).indice = dataCheckListAux.indice;
    ArrdataCheckList(count).codigoDoc = dataCheckListAux.codigoDoc;
   ArrdataCheckList(count).descripcion = dataCheckListAux.descripcion;
   ArrdataCheckList(count).systemanswer =dataCheckListAux.systemanswer;
   ArrdataCheckList(count).answerAllowNo=dataCheckListAux.answerAllowNo;
   ArrdataCheckList(count).deslap = dataCheckListAux.deslap;
   ArrdataCheckList(count).valorEntero = dataCheckListAux.valorEntero;

     Exec Sql Fetch ChkList_Cursor Into: dataCheckListAux;
     check_SQLState();
   EndDo;
   Exec SQL Close ChkList_Cursor;
   Return ArrdataCheckList;

  /End-Free
P                 E
```

La interfaz del procedimiento declara, mediante la palabra reservada del lenguaje RPG ***PI***, que consta de tres argumentos, el primero es el número de la propuesta de Check-List para el desembolso, ***propuestaNumber***, cuyo tipo de dato es un valor numérico con capacidad para diez dígitos, el segundo argumento es el tipo de Check-List a extraer sus datos, ***tipoChkList***, cuyo tipo dato es un valor decimal con capacidad para un digito, y el tercer argumento es el tipo de persona que es el cliente, persona natural o jurídica, ***tipoPersona***, cuyo tipo de datos es carácter de longitud uno. Retorna un arreglo de estructuras de datos de tipo ***ChekList_Data_t*** conteniendo el conjunto de preguntas y respuestas del Check-List consultado, al final verifica que todo haya ido correctamente invocando al servicio ***check_SQLState()***, el cual pertenece al programa de servicio

B01U0001. El controlador **R71M0018** consume el servicio de la forma como lo muestra el siguiente extracto de su código fuente.

```
arrayTipoChkList = getDataChecklist(pRepCre:nTipoExpCL
                                    :dsClienteDto.clasePersona);

Dow arrayTipoChkList(index) <> *Blanks;
    dataTipChkList = arrayTipoChkList(index);
    If  dataTipChkList.answerAllowNo = ANSWER_ALLOW_NO;
       arrAnswerAllowNo(index) = dataTipChkList.codigoDoc;
    EndIf;
    If (prevCategory <> dataTipChkList.kindCategory);
       ncon1 = ncon1 + 1;
       iFlgSN = true;
       WASPECTO = dataTipChkList.deslap;
       WCODDOC = 0;
       Write DATA01;
       ncon1 = ncon1 + 1;
       iFlgSN = false;
       WASPECTO = %char(dataTipChkList.indice) + '.'
                  + %trim(dataTipChkList.descripcion);
       WEXPEDIEN = %char(dataTipChkList.systemanswer);
       WCODDOC = dataTipChkList.codigoDoc;
       Write DATA01;
    Else;
       ncon1 = ncon1 + 1;
       iFlgSN = false;
       WASPECTO = %char(dataTipChkList.indice) + '.'
                  + %trim(dataTipChkList.descripcion);
       WEXPEDIEN = %char(dataTipChkList.systemanswer);
       WCODDOC = dataTipChkList.codigoDoc;
       Write DATA01;
    EndIf;
    prevCategory = dataTipChkList.kindCategory;
    index += 1;
 EndDo;
```

10. Servicio existExpedCheckListProp

El servicio **existExpedCheckListProp** encapsula la implementación de verificar la existencia de un expediente de Check-List asociado a una propuesta de desembolso de algún crédito de acuerdo con su tipo. Este servicio se encuentra implementando dentro del módulo **M07U0002**, el cual forma parte del programa de servicio **B07U0002**, es este último componente a través del cual logra convertirse en un servicio para que

luego pueda ser consumido por los programas clientes que lo necesite, esto se logra mediante el mismo procedimiento aplicado a los primeros servicios que ya se han expuesto en esta sección, ya que siempre se siguen lo mismos pasos para todos lo subprocedimientos que integran un módulo, y que luego llegan a formar parte de un programa de servicio. El código fuente de su implementación se muestra a continuación.

```
 * existExpedCheckListProp() : Verificar la existencia de un  *
 *                             expediente de Check-List       *
 * numeroPropuesta(input) : número de propuesta               *
 * tipoCheckList(input) : tipo de checklist                   *
 * estadoCheckList(input) : estado del checklist 0 ó 1        *
 * return(output) :  True si existe, False caso contrario     *
 ************************************************************
P existExpedCheckListProp...
p                 B                   Export
d                 PI              N
d  numeroPropuesta...
d                                19P 0 Const
d  tipoCheckList                  1P 0 Const
d  estadoCheckList...
d                                 1P 0 Const
d  existExp       s                N   Inz(*off)
d  countExpedts   s               14P 0 Inz(0)
d
 /FREE
   Exec Sql
     Set :countExpedts = (select count(*) as total from F8054
                          where PROPNRO = :numeroPropuesta
                          and TIPCHK = :tipoCheckList
                          and  FLGEST = :estadoCheckList);
    check_SQLState(); //verificamos si todo va bien
    If ( countExpedts > *Zero);
       existExp = *On;
    EndIf;
    Return existExp;

 /End-Free
 p                 E
```

La interfaz del servicio declara, mediante el uso de palabra clave del lenguaje RPG, ***PI***, que recibe tres argumentos, el primero de ellos es el número de la propuesta de Check-List, ***numeroPropuesta***, cuyo tipo de dato es un valor numérico con capacidad para almacenar 19 dígitos, el

segundo argumento es el tipo de Check-List, es decir, previo o durante el desembolso, su tipo de dato es un valor numérico con capacidad para la almacenar un solo dígito, y tercer argumento es el estado de Check-List ***estadoCheckList***, cuyo tipo dato es el mismo que el del segundo argumento. Retorna un valor booleano, ***existExp***, verdadero si todo ha ido correctamente, falso en caso contrario. Al final del procedimiento invoca al servicio ***check_SQLState()***, el cual pertenece al programa de servicio **B01U0001**. El controlador **R71M0018** consume el servicio de la forma como lo muestra el siguiente extracto de su código fuente.

```
If (Not existExpedCheckListProp(pRepCre:nTipoCL:1));
  tablaDigide=PARAM_TABLE;
  digide = DIGIDE_PARAM;
  codTabla=DIGIDE_TABLE;
  schema = getDesLarFromDigide(tablaDigide:digide:codTabla);
  Monitor;
           addLibLE(%trim(schema));
  On-Error;
  EndMon;
```

11. Servicio getNextValueFromCorrelativos

El servicio **getNextValueFromCorrelativos** encapsula la implementación para la obtención del código para expediente de Check-List, ya que este es un valor numérico que se genera de acuerdo con el último valor que se haya generado previamente. El último valor es guardado en una tabla de parámetros para luego ser recuperado y actualizarlo cada vez que es invocado el servicio. Este se encuentra implementado el módulo **M07U0002**, el cual forma parte del programa de servicio **B07U0002**. Y como el resto de servicio de esta sección, es mediante este componente que se convierte en un servicio para que se puede ser invocado por programas

clientes que lo necesiten. El código fuente de su implementación se muestra a continuación.

```
 //*************************************************
 //getNextValueFromCorrelativos(): Obtiene siguiente  *
 //            correlativo para código Check-List     *
 //codigo(input) : último código de Check-List        *
 //flagUpdateValue(input) : Bandera que que indica si *
 //            se actualiza dicho para parámetro      *
 //*************************************************
p getNextValueFromCorrelativos...
p                 B                   Export
d                 PI            19P 0
d  codigo                       19P 0 Const
d  flagUpdateValue...
d                                 N   Const
d
d  nextValue      s             19P 0
d
  /Free

    Exec Sql
      Set :nextValue = ( SELECT (TABCORRELATIVO +1 ) AS VALOR
                         FROM CORRELATIVOS
                         WHERE TABCODIGO = :codigo );

      If (flagUpdateValue);
         updateCorrelativos(codigo:nextValue);
      EndIf;

      return nextValue;
  /End-Free
p                 E
```

El procedimiento declara en su interfaz, mediante la palabra reservada ***PI*** del lenguaje de programación RPG, que recibe dos argumentos de entrada, el primero de ellos es código de Check-List ***codigo***, cuyo tipo de dato es un valor numérico con capacidad para almacenar 19 dígitos, el segundo argumento es un valor booleano, ***flagUpdateValue***, el cual indica que, si desea actualizar el correlativo a su siguiente valor si este es verdadero, el cual para ello inmediatamente invoca al servicio ***updateCorrelativos***. El controlador **R71M0018** invoca este servicio de la manera como lo muestra su siguiente extracto de código.

```
nCodExp = getNextValueFromCorrelativos(tabCodigo:updateCorrelat);
  //Fill structure's data
dsExpChkListPro.COEXPRO = nCodExp;
dsExpChkListPro.PROPNRO = pRepCre;
dsExpChkListPro.NUMDES = pNumDes;
dsExpChkListPro.TIPCHK = nTipoCL;
```

12. Servicio saveExpedCheckListPropuesta

El servicio **saveExpedCheckListPropuesta** encapsula la implementación de la regla del negocio para el registro de un Check-List asociado a una propuesta de desembolso de crédito a la base de datos. Este pertenece al módulo **M07U0002**, el cual forma parte del programa de servicio **B07U0002**, este componente ya se ha explicado detalladamente en este trabajo. Es importante mencionar esto último, ya que es mediante este último componente que el subprocedimiento explicado aquí se convierte en un servicio el cual puede ser consumido por otros programas clientes. El procedimiento para lograr esto es el mismo aplicado para los primeros servicios que se han expuesto detalladamente en esta sección. Su código fuente es el que se expone a continuación.

```
*************************************************************
* saveExpedCheckListPropuesta() : Registra un Expediente de  *
*                                 Check-List.                *
* rowExpedChkListPropuesta(input) : estructura con los datos *
*                                   a registrar              *
* isOk(Output) : Flag, True operación correcta, False        *
*                 caso contrario                             *
*************************************************************
P saveExpedCheckListPropuesta...
P                 B                   Export
d                 PI              N
d rowExpedChkListProP...
d                                     likeDs(rowExpedChkListPropuesta)
d isOk            s               N   Inz(*Off)
d
  /Free
     Exec Sql
     Insert Into F8054 Values(:rowExpedChkListProP);
     check_SQLState();
```

```
     isOk = *on;
     Return isOk;

   /End-Free
P                 E
```

En su interfaz el procedimiento declara en su interfaz, mediante la palabra reservada del lenguaje RPG ***PI***, que recibe un argumento de entrada, este es una estructura de datos ***rowExpedChkListProP***, la cual es de tipo **rowExpedChkListPropuesta**, esta contiene todos los datos necesarios para registrar un expediente de Check-List, retorna en su llamada un valor booleano, ***isOk***, el cual es verdadero si todo ha se ejecutó correctamente o false en caso un error surgió y la operación no se completó. Finalmente invoca al servicio ***check_SQLState()***, del programa de servicio **M01U0001** para verificar que todo haya ido correctamente. El controlador **R71M0018** invoca este servicio para sus tareas de mantenimiento, y la forma que lo hace se muestra en el siguiente extracto de su código fuente y trabaja en conjunción con otros servicios que también se pueden apreciar aquí.

```
     nCodExp = getNextValueFromCorrelativos(tabCodigo:updateCorrelat);
       //Fill structure's data
       dsExpChkListPro.COEXPRO = nCodExp;
       dsExpChkListPro.PROPNRO = pRepCre;
       dsExpChkListPro.NUMDES = pNumDes;
       dsExpChkListPro.TIPCHK = nTipoCL;
       dsExpChkListPro.FLGSIT = 0;
       dsExpChkListPro.FLGEST = 1;
       dsExpChkListPro.USRREG = systemUser;
       dsExpChkListPro.FECREG = zFecRem;

   saveExpedCheckListPropuesta(dsExpChkListPro);
```

13. Servicio getExpedCheckListDto

El servicio **getExpedCheckListDto** encapsula la obtención de los datos más importantes de un expediente de Check-List asociado a una propuesta

de desembolso de crédito. Este servicio extrae los datos más importantes de un registro de la base de datos de un expediente de Check-List, aquí se aplica el patrón de diseño Data Transfer Object, tal como afirma Martin Fowler, este se ajusta a las necesidades que tiene un programa cliente en particular de ciertos datos del modelo **(Fowler, et al., 2011, p. 402)**, para este caso en particular, es necesario presentar los datos extraídos en la interfaz de usuario. Este servicio se encuentra implementado dentro del módulo **M07U0002**, el cual forma del programa de servicio **B07U0002**, es mediante es último componente que se llega a convertirse en un servicio que puede ser consumido por cualquier programa cliente que lo necesite. El procedimiento para lograr esto es el mismo explicado para los primeros servicios expuestos en este trabajo. El código fuente de su implementación se expone a continuación.

```
*****************************************************************
* getExpedCheckListDto() : retorna una estructura de datos con *
*                        la información necesaria para un       *
*                        expediente de Check-List.              *
* nroPropuesta(input) : número de propuesta.                    *
* tipoCheckListPropuesta(input) : Tipo de Check-List.           *
*****************************************************************
p getExpedCheckListDto...
p                 B                   Export
d                 PI                  likeDs(expedChkListPropuestaDto)
d nroPropuesta                  19P 0 Const
d tipoCheckListPropuesta...
d                                1P 0 Const
d dsExpedCkListDto...
d                 ds                  likeDs(expedChkListPropuestaDto)
  /Free

    Exec Sql
       Select COEXPRO,FLGSIT,COALESCE(COMREV,'') Into :dsExpedCkListDto
       From F8054 Where PROPNRO = :nroPropuesta
       And TIPCHK = :tipoCheckListPropuesta;
       check_SQLState();

     Return dsExpedCkListDto;
  /End-Free
```

```
p                   E
```

El servicio declara en su interfaz, mediante la palabra reservada **PI** del lenguaje **RPG**, que recibe dos argumentos, el primero de ellos es el número de la propuesta de Check-List ***nroPropuesta***, cuyo tipo de dato es un valor número con capacidad para almacenar 19 dígitos, el segundo es el tipo de expediente de Check-List, previo o durante un desembolso, cuyo tipo de dato es un valor numérico con capacidad para almacenar un solo digito. El controlador **R71M0018** invoca a este servicio de la manera como lo muestra el siguiente extracto de su código fuente.

```
Else;
   expChkListDto = getExpedCheckListDto(pRepCre:nTipoCL);
   nCodExp = expChkListDto.codigoProuesta;
   nFlgSit = expChkListDto.flagSituacion;
   sAntesComen = expChkListDto.comentario;
EndIf;
```

14. Servicio check_SQLState()

El servicio check_**SQLState** es una utilidad de uso general para manejar errores de los programas RPG que utilicen sentencias SQL, tales como las que hemos venido exponiendo a lo largo de esta sección. Su tarea es enviar los mensajes de error a los programas clientes que invoquen un servicio en caso de que éste falle, hacer que el programa pare su ejecución y tener un mejor manejo ante esta situación, para ello utiliza la API del sistema operativo ***System i***, **QMHSNDPM (IBM, s.f.)**. Esta mapea el último código de estado de la variable interna ***SQLSTATE***, verifica que sea diferente de cero, lo que significa que ocurrió un error al momento de ejecutar alguna sentencia e inmediatamente envía un mensaje de error a la cola de ejecución del programa llamador. Es una adaptación personalizada

de un trabajo expuesto por un autor de unas de las fuentes citadas, Paul Tuohy **(Tu**ohy, s.f.), que este desarrolló para tratar con este tipo de situaciones, ya que en el original este no se encuentra dentro de un componente que sea de tipo ***SRVPGM (Diedrich, y otros, 2016, pág. 78)**.

Este servicio se encuentra implementado dentro de módulo **M01U0001** el cual pertenece al programa de servicio **B01U0001**, el cual ya se ha expuesto con detalle en este trabajo, y es mediante este componente que se convierte en un servicio. Este servicio no es invocado dentro del controlador **R71M0018** el cual estamos tratando, sino que es invocado dentro de los procedimientos del módulo **M07U0002** los cuales manejan sentencias SQL en su codificación, esto es posible ya que ambos están instalados en el directorio de enlace **SRVPBND** y el programa de servicio **M07U0002** referencia a este directorio mediante la palabra reservada **BNDDIR** dentro de su codificación. Se muestra a continuación el código fuente de su implementación, y la forma cómo usarlo ya se ha visto en todos los servicios que se han expuesto, por lo tanto, ya no es necesario hacerlo.

```
P check_SQLState  B                   export
D                 PI              n
  // Standard API Error Data Structure
d APIError        DS                  qualified
d  bytesProvided               10i 0 inz(%size(APIError))
d  bytesAvail                  10i 0 inz(0)
d  msgId                        7a
d                               1a
d  msgData                    240a

   // Prototype for QMHSNDPM (Send Program Message) API
d sendProgramMessage...
d                 PR                  extPgm('QMHSNDPM')
d   messageID                   7a    const
```

```
d   messageFile                 20a   const
d   messageData                256a   const
d   messageDataLength...
d                               10i 0 const
d   messageType                 10a   const
d   callStackEntry...
d                               10a   const
d   callStackCount...
d                               10i 0 const
d   messageKey                   4a
d   errorCode                         likeds(APIError)

    // Work fields
d messageKey      s              4a
d messageType     s             10a
d messageText     s           1024a
d status          s              n

d                 DS
d  lastState                     5a
d  status_SQL                    2a   overLay(lastState)

 // Constants
d W_DIAGNOSTIC    C                   '*DIAG'
d W_EOF           C                   '02'
d W_ESCAPE        C                   '*ESCAPE'
d W_MSGF          C                   'QCPFMSG   *LIBL'
d W_MSGID         C                   'CPF9897'
d W_STACK_ENTRY   C                   '*'
d W_STACK_COUNT1  C                   1
d W_SUCCESS       C                   '00'
d W_WARNING       C                   '01'

  /FREE
    // Get last state
    exec SQL
       get diagnostics condition 1 :lastState = RETURNED_SQLSTATE;

       //All OK - just return
      if (status_SQL = W_SUCCESS);

        // EOF - return true - but no message
       elseIf (status_SQL = W_EOF);
         status = *on;

         //Warning - send Diagnostic message
       elseIf (status_SQL = W_WARNING);
         messageType = W_DIAGNOSTIC;
         exec SQL
           get diagnostics condition 1: messageText = MESSAGE_TEXT;

         //Anything else - Send a Escape message
       else;
         messageType = W_ESCAPE;
         exec SQL
          get diagnostics condition 1: messageText = MESSAGE_TEXT;
        status = *on;
      endif;
```

```
     if (messageType <> *blanks);
         dump(A);
         messageText = lastState + ' ' + messageText;
         sendProgramMessage( W_MSGID
                           : W_MSGF
                           : messageText
                           : %len(%trimr(messageText))
                           : messageType
                           : W_STACK_ENTRY
                           : W_STACK_COUNT1
                           : messageKey
                           : APIError);
     endif;
     return status;

  /END-FREE
p                 E
```

Capítulo 5: Análisis y revisión de resultados finales

5.1 Reporte de resultados de la aplicación de marco de trabajo propuesto e interpretación

En este capítulo se van a exponer las ventajas que representa aplicar el Marco Arquitectónico para el desarrollo de aplicaciones empresariales propuesto en este trabajo de tesis utilizando las herramientas que proporciona el lenguaje de programación estructurado **RPGLE**, nos referimos al modelo de desarrollo de programación denominado **ILE**.

En aplicaciones construidas usando el lenguaje de programación estructurado **RPGLE** empleando el antiguo modelo de programación denominado **OPM**, no permite emplear las modernas técnicas de desarrollo de software, la aplicación de prácticas estándar, el uso patrones arquitectónicos, el aprovechamiento

máximo de todas las nuevas características que brinda el nuevo modelo de desarrollo denominado **ILE** que ofrece el lenguaje RPGLE, en fin podría decirse el uso y aplicación de todas las herramientas que la ingeniería de software ha construido a lo largo de este tiempo.

La creación de aplicaciones que emplean el modelo antiguo de desarrollo de software **OPM** se les denomina aplicaciones monolíticas, ya que consta de un solo programa, el cual es una gran unidad de compilación, en este se encuentran implementadas las reglas de negocio, el acceso al modelo o data de la aplicación y la interacción con la interfaz de usuario. No existe pues el sentido de separación de responsabilidades con el empleo de esta arquitectura. Este modelo antiguo es el empleado en aplicaciones heredadas o también llamadas ***LEGACY APPLICATIONS***, y es este el escenario ante el cual nos hemos encontrado para proponer un nuevo enfoque de trabajo que permita mejorar el desarrollo de aplicaciones empresariales proponiendo un nuevo marco de trabajo para su construcción.

Toda aplicación de software no emplea un único patrón de diseño para su construcción, si no que por el contrario y para lograr una mayor escalabilidad, emplea muchos patrones arquitectónicos y otras técnicas de la ingeniería de software. El nuevo marco de trabajo ha empleado para la arquitectura de las aplicaciones empresariales, usando el lenguaje de programación estructurado RPGLE, tres patrones arquitectónicos de diseño adecuándolos al contexto de la tecnología con la cual se está trabajando, ya que al ser el lenguaje un elemento que no se puede reemplazar este se convierte en un componente arquitectónico. Se resalta el hecho de ser un paradigma de programación estructurado ya que tiene sus restricciones respecto a los nuevos paradigmas de programación

recientes creados para tratar con una entidad tan cambiante como lo es el negocio de la empresa para la cual se construye un sistema software.

Los patrones de diseño arquitectónicos empleados y adaptados han sido los siguientes, el patrón Data Transfer Object (**DTO)** por sus siglas en inglés, el cual permite transferir entre diferentes partes de la aplicación sólo la data importante para estas y nada más. Además, se han empleado dos arquitecturas estándar muy usadas en la construcción de modernas aplicaciones empresariales, la arquitectura basada en capas, Modelo Vista Controlador (**MVC**) y la arquitectura orientada a servicios (**SOA**), ambas arquitecturas permiten separar la construcción de la aplicación en diferentes capas cuyo principal objetivo es maximizar la reusabilidad de código fuente y minimizar el impacto de cambios y mantenimiento de la aplicación.

Es posible también integrar ambos modelos de programación ofrecidos por la tecnología **RPG**, es decir, el antiguo modelo **OPM** y el nuevo modelo **ILE**, esto se ha demostrado a lo largo del desarrollo del desarrollo de esta tesis explicando detalladamente a través de un nuevo desarrollo la manera de cómo integrarlo y hacer convivir ambas arquitecturas sin necesidad de reconstruir toda la aplicación desde cero, ya que una de las restricciones es que la nueva aplicación debe de formar parte del software CORE del negocio existente, pues este soporta muchos procesos de negocios de la empresa. Sin embargo, es partir de la construcción de esta nueva aplicación que se plantea para futuros desarrollos el empleo de la nueva arquitectura propuesta con la finalidad de ir modernizando la construcción de aplicaciones empresariales que hagan uso de la tecnología **RPGLE** aplicando los conceptos de ILE y el uso de modernas técnicas de arquitectura de software.

El empleo de todas las técnicas descritas permiten diseñar aplicaciones modulares lo cual lleva a hacer una comparación con aplicaciones monolíticas que es el modelo tradicional con el cual se ha contrastado el presente trabajo, así es que a continuación se ofrece un cuadro comparativo de ambos que nos dan idea de las ventajas y desventajas de cada uno de ellos, así como la comprobación estadística de la complejidad ciclomática que permite medir el nivel de complejidad de un programa lo cual proporciona una medida para saber cuan fácil es testear, entender y mantener una sección de código fuente de la aplicación **(Amra, y otros, 2014)**.

El análisis de los resultados obtenidos para la medición de la medición de la complejidad ciclomática se llevó a cabo empleando el análisis de varianza de dos factores con varias muestras por funcionalidad por cada arquitectura expuesta en este trabajo.

La arquitectura monolítica que usa el modelo de programación OPM del lenguaje RPG hace uso de subrutinas para implementar una funcionalidad, en contraste con la arquitectura propuesta, en la cual se plantea un servicio para cada funcionalidad que se requiera de la aplicación, se han tomado las muestras de ambas arquitecturas y se ha procedido a su tratamiento estadístico, la tabla 5.3 muestra el valor de la complejidad ciclomática por cada funcionalidad en cada arquitectura. De acuerdo con acuerdo con la teoría este valor se interpreta de la siguiente manera : un valor del 1-10 se interpreta como programa simple, sin mucho riesgo, un valor del 11-20 es un programa más complejo cuyo riesgo es moderado, un valor de 21-50 quiere decir un programa complejo de alto riesgo y finalmente un valor de 50 significa que el programa es no testeable cuyo riesgo es muy alto.

Tabla 5.1

Datos para el análisis estadístico de resultados

Ítem	**Descripción**
Factor A:	Complejidad ciclomática de arquitectura monolítica
Factor B:	Complejidad ciclomática de arquitectura propuesta.
Variable respuesta o Variable dependiente (Efecto) :	Valor de complejidad ciclomática
Número de niveles del factor A:	a = 2
Número de niveles del factor B:	b = 8
N° total de observaciones realizadas	16
Diseño experimental utilizado para el análisis:	Diseño Bifactorial
Nivel de significancia (alfa):	5%

Elaboración Propia

Prueba de Hipótesis: En la tabla 5.2 se muestra la prueba de hipótesis que se utilizo para el análisis estadístico de los resultados obtenidos.

Tabla 5.2

Respecto al efecto de los factores principales

Factores	**Hipótesis**
Aplicación del nuevo marco de trabajo arquitectónico.	**H_0** : No existe diferencia significativa en el valor de la COMPLEJIDAD CICLOMÁTICA DEL NUEVO MARCO ARQUITECTÓNICO PROPUESTO, es

	decir, es efecto es igual a cero. **H_1:** Existe diferencia signifativa en el valor de la COMPLEJIDAD CICLOMÁTICA DEL NUEVO MARCO DE TRABAJO PROPUESTO, es decir, cuando se aplica el nuevo marco de trabajo arquitectónico propuesto su efecto es diferente de cero.

Elaboración propia

Conclusión: cuando el valor de "F calculado" es mayor que el "F de tabla", se rechaza **H_0**.

A continuación se muestra el análisis varianza (**ANOVA**) de la complejidad ciclomática por cada funcionalidad en ambas arquitecturas – Cantidad de funcionalidades: 16

Tabla 5.3

Tabla de complejidad ciclomática de los marcos arquitectónicos analizados.

Arquitectura monolítica		**Marco Arquitectónico propuesto**	
Funcionalidad	**Complejidad ciclomática**	**Funcionalidad**	**Complejidad ciclomática**
sbr002	9	getDataChecklist	2
sbr006	15	saveTipoExpedCheckList	2
sbr008	13	deleteTipoExpedCheckList	2
sbr001	5	saveExpedCheckListPropuesta	2
sbr012	5	existExpedCheckListProp	3
sbr015	3	getExpedCheckListDto	2

sbrAdmChkList	14	getNextValueFromCorrelativos	2
Main	35	getDesLarFromDigide	3

Elaboración Propia.

Fuente: Guía de observación.

Tabla 5.4

Análisis de la varianza de la complejidad cíclica de una aplicación empleando el nuevo marco de trabajo.

FUENTE DE VARIACIÓN	**GRADOS DE LIBERTAD**	**SC**	**MC**	**F CALCULADO**	**F TABLA ALFA = 5%**	**Conclusión**
Complejidad ciclomática	1	495.0625	495.0625	9.476499744	4.60010994	Se reschaza H_0
Error	14	731.375	52.24107143			
Total	15	1226.4375				

Elaboración Propia

Fuente: Análisis de Varianza (ANOVA)

En la **tabla 5.4** se muestra el análisis de la varianza para la complejidad ciclomática de las funcionalidades de ambas arquitecturas, donde se busca demostrar si el uso del nuevo marco arquitectónico de trabajo propuesto afecta significativamente a la complejidad cíclica de una aplicación.

En dicha tabla se observa que al 5% de nivel de significancia, el uso del nuevo marco arquitectónico propuesto (F_C = 9.476499744), actuando de manera

independiente, afecta significativamente la complejidad ciclomática de una aplicación. Con lo que se concluye que se rechaza la Hipótesis nula planteada en la tabla 5.2.

Tabla 5.5

Comparación de arquitecturas : Monolítica y Modular

Característica	Modular	Monolítica
Fácil de cambiar	Reduce la dificultad, costo y tiempo para realizar cambios a la aplicación.	La complejidad es muy elevada ya que todas las funciones están implementadas en un solo módulo, requiere mayor esfuerzo y tomar un riesgo mayor del esperado.
Fácil de entender	En una arquitectura modular se puede empezar por comprender sólo una parte de la aplicación, aquella donde se requieren hacer los cambios. No es necesario comprenderla	Se debe comprender toda la aplicación a la vez, ya que el componente más pequeño es la aplicación entera. Es difícil enfocarse en una sola parte de esta, ya que la ausencia de capas y

	en su totalidad.	separación de responsabilidades obliga a entenderla toda.
Fácil de testear	Es fácil testear una aplicación modular, ya que puedes enfocarte en un módulo en particular y escribir test unitarios para este. Los test unitarios son recursos valiosos para validar el código fuente de la aplicación.	Las arquitecturas monolíticas no permiten la opción hacer test unitarios. Debido a su complejidad alta es muy difícil poder probar ciertas partes de la aplicación. Esto puede llevar el riesgo de insertar bugs en la aplicación.
Reusabilidad	La arquitectura modular se enfoca en crear una aplicación como la composición de pequeñas piezas. Estás piezas construidas con un buen diseño pueden ser reutilizadas en distintas partes de la aplicación.	Al ser la aplicación un gran componente, no se puede reutilizar ninguna parte de la aplicación. Si se quiere hacer esto, la opción es copiar y pegar el código fuente de la funcionalidad en cada parte de la aplicación dónde se le necesite.
Responsabilidad única	Las responsabilidades y	No existe este concepto

	funcionalidades están repartidas entre todos los módulos que componen la aplicación.	en una arquitectura monolítica al estar todas las funcionalidades en un solo modulo.
Alta cohesión	Las funcionalidades que están relacionadas se encuentran en un solo módulo y separadas de aquellas con las cuales no tienen relación. La arquitectura modular permite hacer este tipo de diseño.	No existe en absoluto el concepto de cohesión en una arquitectura monolítica ya que todas las funcionalidades, incluso aquellas que no guardan relación se encuentran en solo módulo.
Bajo acoplamiento	Baja dependencia entre los módulos que componen la aplicación. La arquitectura modular permite crear diseños en los cuales un componente no sea demasiado dependiente de otro.	La dependencia no existe en este caso, las arquitecturas monolíticas sólo cuentan con un componente que es la aplicación entera en si misma.

Comparación de características de las arquitecturas de software.

5.2 Discusión de resultados

Después de haber aplicado una nueva arquitectura al desarrollo de aplicaciones empresariales usando el lenguaje de programación estructurado RPGLE se pueden obtener los siguientes aspectos importantes que cabe resaltar :

El nuevo modelo de programación que proporciona el lenguaje de programación **RPGLE** denominado **ILE**, dota a esta tecnología con la capacidad para aplicar las modernas técnicas de la arquitectura de software. Este nuevo modelo aporta las herramientas necesarias para dotar al arquitecto de software con la capacidad de crear mejores diseños para el desarrollo de aplicaciones empresariales, ya que es justo el aspecto cambiante de los negocios que hace que las aplicaciones sean construidas pensando en la adaptabilidad y capacidad de cambios que el negocio sufre constantemente y que deben ser reflejados de forma inmediata en el sistema software que soporta el negocio.

Esto es posible por qué el nuevo modelo de programación cuenta con conceptos como los programas de servicio, módulos, directorios de enlace, subprocedimientos, lenguaje de enlazado, grupos de activación **(Diedrich, y otros, 2016, pág. 78)**. Estas herramientas proporcionan un marco de trabajo para poder hacer mejores diseños para la construcción de aplicaciones empresariales, especialmente las arquitecturas que se plantean para estas. Es importante señalar que este nuevo modelo de programación permite integrar diferentes tipos de tecnologías como lo son Java, PHP, Python, C, C++ entre otras y pueden convivir para lograr una mejor escalabilidad e interoperabilidad con sistemas externos a la aplicación, como son los middlewares.

La nueva arquitectura y marco de trabajo planteados en esta tesis requiere un mayor esfuerzo en la fase de diseño y la comprensión correcta de todos los

conceptos del nuevo modelo de programación **ILE**, lo cual no es tarea fácil, como se ha expuesto muchos de estos conceptos no son entendidos por muchos programadores que los encuentran complicados, lo que les lleva a seguir empleando el modelo antiguo para el desarrollo de aplicaciones **(Klement, presentations, pág. 3)**, es decir, crear arquitecturas monolíticas sin usar ninguna técnica moderna de desarrollo de software.

Se puede decir que, no es excusa el encontrar una aplicación antigua y seguir usando el antiguo modelo de desarrollo para la construcción de nuevas aplicaciones usando un lenguaje de programación estructurado como lo es RPG, es posible aprovechar las nuevas herramientas que se han explicado e integrar la nueva arquitectura y la antigua, y hacer que ambas interactúen sin que esto represente un problema a la vez que en paralelo se puede ir armando un plan de modernización de toda la aplicación en general con el fin de sacar el máximo provecho del nuevo modelo de programación **ILE**.

Conclusiones

- Se pudo crear una nueva arquitectura de software basada en capas aplicando las modernas técnicas de arquitectura de software, usando el modelo de programación ILE que ofrece el lenguaje de programación estructurado RPG para la construcción de aplicaciones empresariales.
- Se pudo construir una nueva aplicación software empresarial usando la tecnología IBM empleando para ello la nueva arquitectura y diseño propuestos en este trabajo, esto se demostró con la implementación del requerimiento normativo de la SBS, el cual se le denominó Check-List.
- Se demostró las ventajas que representa usar la nueva arquitectura propuesta con respecto a la anterior, la cual es una arquitectura monolítica empleada por las aplicaciones antiguas o también llamadas LEGACY APPLICATIONS. Esta nueva arquitectura muestra ventajas tales como: diseño modular, arquitectura basada en capas y modelado de negocio basado en objetos, esto último se logra mediante el uso de estructuras de datos.
- Es posible integrar dos modelos arquitectónicos distintos y que estos convivan sin conflictos : aplicaciones de arquitecturas monolíticas y aplicaciones construidas con arquitecturas multicapas.

Recomendaciones

- Ampliar el estudio de patrones arquitectónicos y encontrar la forma de aplicarlos y adaptarlos empleando el lenguaje de programación estructurado RPG.
- Probar la característica del modelo de programación ILE del lenguaje RPG y su integración con diferentes tipos de tecnologías como lo son Java, C, C++, Python o PHP construyendo para ello una nueva aplicación que haga uso de las diferentes tecnologías mencionadas.
- Proponer la nueva arquitectura que se expone en este trabajo de tesis a empresas que usen la tecnología RPG, empezando con una prueba de concepto que demuestre su viabilidad y adaptación a los desarrollos con los que ya cuenta la empresa.
- Dar a conocer en ambientes académicos la tecnología RPG cuyo uso es muy común en empresas del sector financiero de la región Piura y del Perú, y de esta forma beneficiar a las empresas y los estudiantes universitarios para quienes representa una gran oportunidad laboral en el futuro.

Referencias bibliograficas

Amra, N., Bedoya, H., Caims, T., Criukshank, D., Diedrich, R., Eberhard, J., . . . Woodbury, C. (2014). *Modernizing IBM i Applications from the Database up to the User Interface And Everything in Between.* New York: IBM RedBooks.

Bass, L., Clements, P., & Kazman, P. (2013). *Software Architecture in Practice.* Pearson.

Bedoya, H., Cruz, F., Lema, D., & Singkorapoom, S. (2016). *External Procedures, Triggers, and User-Defined.* IBM.

Descartes, R. (1637). *Discurso del Método.* Tecnos (Grupo Anaya).

Diedrich, R., Diephius, J., Gantner, S., Minette, J., Paris, J., Robinson, K., . . . Tuohy, P. (2016). *Who Knew You Could Do That With RPG IV? Modern RPG for The Modern Programmer*. IBM .

F, E. M. (s.f.). *Métodos y técnicas de investigación.* Trillas .

Fowler, M. (2002). *Design - Who needs an architect?* Obtenido de Design - Who needs an architect?: https://ieeexplore.ieee.org/document/1231144

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). *REFACTORING, IMPROVING THE DESING OF EXISTING CODE.* ADDISON-WESLEY.

Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., & Stafford, R. (2011). *Patterns of Enterprise Application Architecture.* Addison Wesley.

Gorton, I. (2011). *Essential Software Architecture.* New York: Springer.

IBM. (s.f.). *IBM* . Obtenido de IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/apis/QMHSNDPM.htm

Klement, S. (2010). *RPG Programming*. Obtenido de https://www.scottklement.com/presentations/ILE%20Concepts%20(For%20The%20Impatient%20RPG%20Programmer).pdf

Klement, S. (s.f.). *presentations.* Obtenido de RPG Programming: https://www.scottklement.com/presentations/Pattern%20to%20Good%20ILE%20with%20RPG.pdf

Meyers, B., & Buck, J. (2010). *PROGRAMMING in RPG IV.*

Pressman, R. (2010). *Ingeniería del Software, Un enfoque práctico* (SÉPTIMA EDICIÓN ed.). (V. Campos Olguín, & J. Enríquez Brito, Trads.)

Smith, B., Barbeau, M., Gantner, S., Paris, J., Vincetic, Z., & Zupka, V. (2000). *Who Knew You Could Do That With RPG IV? A Sorcerer's Guide to System Access And More.* Minessota: IBM.

The Institute of Electrical and Electronics Engineers, Inc. (2000). *https://www.ieee.org/.* Retrieved from http://cabibbo.dia.uniroma3.it/ids/altrui/ieee1471.pdf

Tuohy, P. (s.f.). *Embedded SQL Exception/Error Handling*. Obtenido de IT Jungle: https://www.itjungle.com/2014/04/02/fhg040214-story01/

Anexos

Anexo A: check-list para sistema de gestión de créditos para aprobación de desembolso.

Formato de Control Interno para Desembolso de Créditos

	ASPECTOS PARA REVISAR	EXPEDIENTE FÍSICO	
	ANTES DEL DESEMBOLSO	**SI**	**NO**
	De la Tarjeta de Información Básica del Solicitante		
1	Copia del DOI del cliente aval(es) y/o garante(s) cónyuge(s)		
2	Vig. de poder, opinión legal favorable por desembolso.		
3	Hoja amarilla con firmas de los participantes de crédito		
4	Firma y sello del Sectorista de crédito en Tarjeta de Información		
5	Firma y Sello del Sectorista. Actualizo datos en sistema SOFIA		
	De la Solicitud de Crédito		
6	Solicitud de Crédito sin enmendaduras, borrones		
7	Firma y sello de Sect. Crédito, y un integrante. del SCAC		
8	Documentación (originales o copias) sustentan el crédito		
	De la Resolución de Créditos		
09	En sección "Observación" ver N°.... propuesta del SGCRED		
10	Firma y sello de dos de los miembros integrantes del SCAC		
11	En Ofic. Informativa una firma integrantedel SCAC		
12	Verificar levantamiento de observ. En solicitud de crédito		
	De las Garantías No Inscribibles (Blandas)		
13	Bs Mb: Contr. Garantía mob. Firmado los participantes del créd.		
14	DD.JJ. de Bs firmada por participantes de crédito.		
15	DD.JJ. Constatación de exist. de Bs. firmada por sectorista.		
16	Garantías con desposesión: Copia de certificado Y		
17	Carta de autorización firmado por propietario de la garantía.		
	De las Garantías Inscribibles (Reales)		
18	Cop. Contrato de garantía mobiliaria pre Const. Legalizados.		
19	Cop. Contrato de garantía mobiliaria Const. Legalizados.		
20	Copia del Testimonio de Constitución de Escritura Pública.		
	DURANTE EL DESEMBOLSO	**Si**	**No**
1	¿El cliente conocía el monto de su crédito?		
2	Indicar si hubo visita al domicilio o negocio del cliente.		
3	Actitud sospechosa del cliente durante desembolso del crédito.		
4	Actitud sospechosa de algún colaborador de la empresa.		
Resultados:			
	Si respondió "SI", proceda a realizar el desembolso (trx. 6525).		
	Si respondió "No": Queda como pendiente		

Anexo B: pase a producción de la aplicación check-list

Nº CONTROL:

PASE A PRODUCCIÓN Nº 013-2015 / NEBAR

CMAC-PIURA S.A. C.	**Pase a Producción**	**AREA DE SISTEMAS**
Referencia	PROYECTO SGCRED: Req. 282-2013	
Área Solicitante	ÁREA DE CRÉDITOS	
Inicio de Tarea	23/10/2015	
Fin de Tarea	16/12/2015	
Analista/Programador	NEBAR	
Nº Referencia	REQ. OYM 282-2013-Adenda	
Momento del Pase	ANTES DEL CIERRE DIARIO.	

2. ARCHIVOS Y/O PROGRAMAS A PASAR

3.

Archivos			**Programas**			**Scripts**
Físicos/Lógicos	**Pantallas**	**Impresión**	**CL**	**RPGLE**	**SQLRGPLE**	**SQL**
NUEVOS						
	P71M0018			R01U0001 (COPY)	R71M0018	1 ALTERTABLE8051.sql
	P71C0018			R07U0001 (COPY)	R71C0018	2 UPDATEF8051.sql
	P71C0190	I71C0190			R71C0190	
	P71C0191	I71C0191			R71C0191	
					R71P0059	
					M01U0001 (MODULO)	
					M07U0002 (MODULO)	

MODIFICADOS						
F8051(sql)						
	P7102310			R7102310		
	P7101504			R7101504		
				R94C0001		
	P7102501			R7102501		
RECOMPILADOS						
				R76C0054		

DESCRIPCION DEL PROCESO

Como se explica en el pase No iseries Req. 282-2013 para el manejo de check list de créditos se ha realizado los ajustes en el Sistema de Gestión de Créditos (SGCRED).

EL proceso de desembolso se requiere CREAR UN CHECK LIST DE OPERACIONES el cual serán llenadas por operaciones y verificadas por los administradores.

Los tipos de Check List de operaciones que se muestran son los siguientes:

PARA SOFIA:

Se ha realizado los siguientes ajustes para el proceso de desembolso de una propuesta:

1. De la transacción 6525, se ha modificado la pantalla P7102310 y el programa R7102310 para agregar una nueva columna CLP que muestra si se ha anexado el check list de operaciones y una opción para que el usuario elija si la el desembolso es presencial o no.
2. Antes de realizar el desembolso se tiene que llenar un check list de operaciones para lo cual se ha creado el programa R71M0018 y la pantalla P71M0018 que muestra un check list tanto previa como durante el desembolso.
3. Para el caso del check list durante el desembolso se tendrá que atender un mensaje para validar el check list, en cual se ha tenido que modificar la pantalla P7102310 y el programa R7102310.
4. Para las consultas de documentos (transacción 3) se ha modificado la pantalla P7101504 y el programa R7101504 donde muestra un F15: Chk List.
5. Para el Menú de supervisor – 5002, se ha agregado las consultas de documentos (12->5), se ha modificado la pantalla P7102501 y el programa R7102501 donde se muestra un nuevo F15:Chk.Lst (Check List).
6. Y para consulta de expedientes check list se ha creado la pantalla P71C0018 y el programa

R71C0018.

7. Para el caso de alertas a los administradores ha modificado el programa R94C0001 y creado el programa R71P0059.
8. Se han creado los programas R71C0190 y R71C0191 para los reportes de las restricciones del CHECK LIST, el primero con restricciones levantas entre fecha, y el segundo de ellos que muestra un reporte de restricciones por levantar entre fechas, el cual es llamado desde el programa R7102313.

Procedimiento a realizar en producción

PASOS A SEGUIR:

1. Ejecutar el siguiente script "**ALTERTABLE8051.sql**" que contempla la creación de tablas del proyecto SGCRED para el manejo del check list en la biblioteca SOFIAD.
2. Ejecutar el siguiente script "**UPDATEF8051.sql**" que contempla la actualización de tablas del proyecto SGCRED para el manejo del check list en la biblioteca SOFIAD
3. Copiar el fuente de programa enlazador **M01U0001** y **M07U0002** al archivo físico de fuentes **QSRVSRC**, de existir dicho fuente, se debe reemplazar.
4. Copiar los archivos fuentes de los programas **R07U0001** y **R07U0001** al archivo físico de fuentes QRPGCPYLE, de existir dichos archivos fuentes se deben reemplazar.
5. Pasar los archivos fuente **M01U0001** y **M07U0002** de desarrollo a la **SIAFF/QRPGMODSRC** de producción.
6. Copiar los objetos **M07U0001** y **M07U0002** de la librería SIAFOX del ambiente desarrollo a la librería SIAFO del ambiente de producción
7. Pasar los objetos **B01U0001** y **B07U0002** de tipo *SRVPGM de al SIAFOX a la SIAFO de producción.
8. Ejecutar el siguiente comando **RMVBNDDIRE BNDDIR(SRVPBND) OBJ((B01U0001))**.
9. Ejecutar el siguiente comando **RMVBNDDIRE BNDDIR(SRVPBND) OBJ((B07U0002)).**
10. Agregar las entradas del programa de servicio al directorio de enlace SRVPBND de SIAFO con el siguiente comando **ADDBNDDIRE BNDDIR(SRVPBND) OBJ(B01U0001)**.
11. Agregar las entradas del programa de servicio al directorio de enlace SRVPBND de SIAFO con el siguiente comando **ADDBNDDIRE BNDDIR(SRVPBND) OBJ(B07U0002).**
12. Área de seguridad y continuidad debe crear una nueva opción para el gerente regional, jefe de créditos de Empresa Financiera, Administrador de agencia y supervisor de operación en la siguiente sección:
    - ✓ 05 Colocaciones
    - ✓ 51 Opciones generales
    - ✓ 20 Clientes con restricciones levantadas

COD.	**OPC.**	**TEXTO**	**TIT**	**MIL.**	**SUBMENU**	**PROGRAMA**	**PAR.**	**COMANDO**
XX XX	X X	Clientes con restricciones levantas entre	-	1	-	**R71C0190**	06	

		fechas						

Todos los programas se encuentran en la Biblioteca NEBAR.

Fecha de Entrega	16/11/2015

ANALISTA / PROGRAMADOR	**AGENTE DE CALIDAD**
SUBJEFE DE DESARROLLO	**PRODUCCIÓN**

Anexo C: pruebas unitarias enviadas al área de calidad

REQ.282-2013/NEBAR
Pruebas unitarias – SOFIA

Nombre Incidente/ Requerimiento/Proyecto	REQ.282-2013				
Desarrollador(s):	Nelson Abel Barranzuela Iman				
Fecha Pruebas :	09/12/2015	**Hora Inicio :**	8:00 am	**Hora Fin :**	6:45 pm
Fecha de Entrega :	16/12/2015				

N°	Requerimiento	Estado
1	Automatización de Check List en SGCRED y SOFIA, modificaciones al requerimiento original.	

Comentario

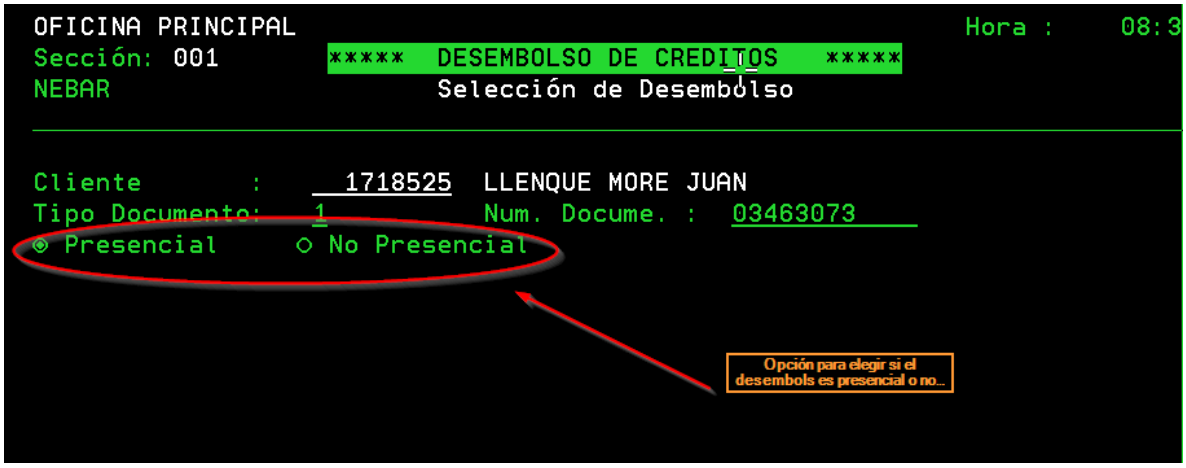
En el documento solo se hace seguimiento de pruebas en la parte del SOFIA. Aquí se consideran todas las observaciones que se le han hecho al requerimiento original, la mayoría son validaciones que aplican para las personas naturales y jurídicas, así es que se solicita que el usuario responsable de las pruebas verifica tales validaciones.

Procedimiento

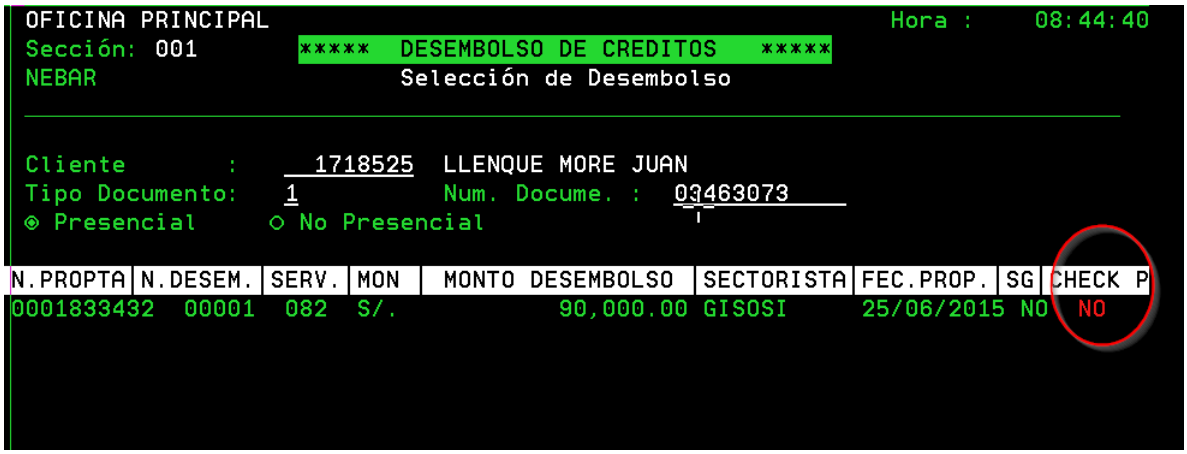
EL proceso de desembolso se requiere CREAR UN CHECK LIST DE OPERACIONES el cual serán llenadas por operaciones y verificadas por los administradores.
PARA SOFIA:
Se ha realizado los siguientes ajustes para el proceso de desembolso de una propuesta:

1. De la transacción 6525, se ha modificado la pantalla P7102310 y el programa R7102310 para agregar una nueva columna CLP que muestra si se ha anexado el check list de operaciones.
2. Antes de realizar el desembolso se tiene que llenar un check list de operaciones para lo cual se ha creado el programa R71M0018 y la pantalla P71M0018 que muestra un check list tanto previa como durante el desembolso.
3. Para el caso del check list durante el desembolso se tendrá que atender un mensaje para validar el check list, en cual se ha tenido que modificar la pantalla P7102310 y el programa R7102310.
4. Para las consultas de documentos (transacción 3) se ha modificado la pantalla P7101504 y el programa R7101504 donde muestra un nuevo menú que se llama F15: Chk List
5. Y para consulta de expedientes check list se ha creado la pantalla P71C0018 y el programa R71C0018.
6. Para el caso de alertas a los administradores ha modificado el programa R94C0001 y creado el programa R71P0059.

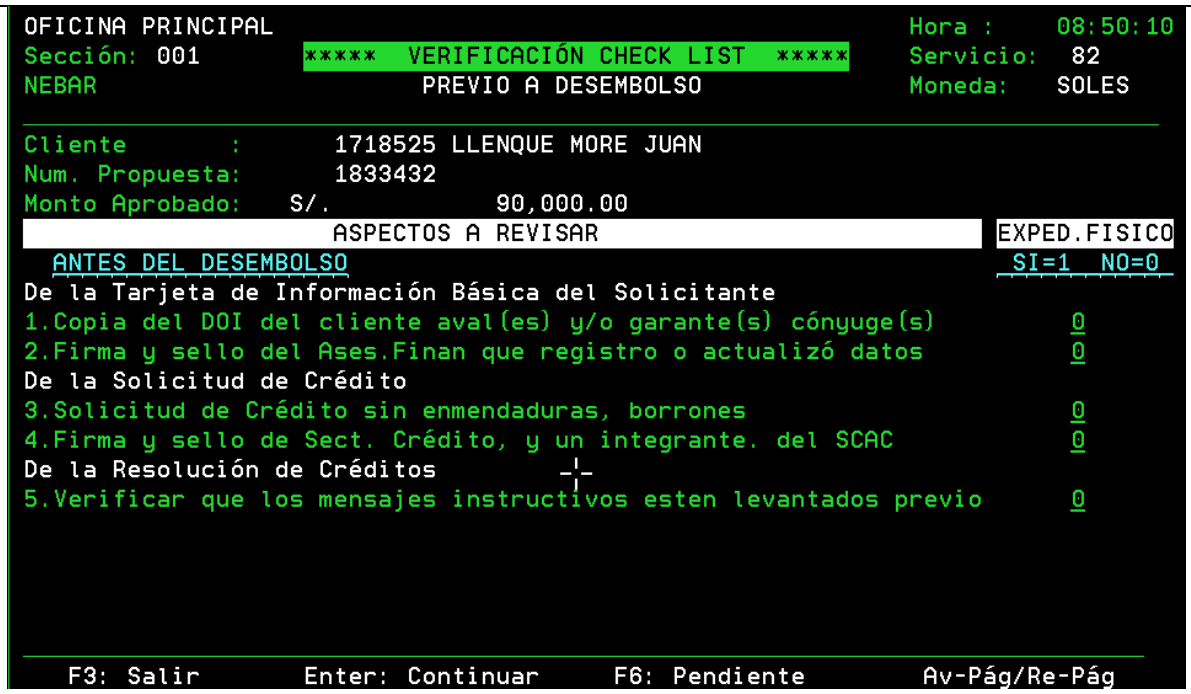
MENU OPERACIONES: TRANSACCIÓN 6525



1) **Ingresamos los datos del cliente que tenga una propuesta con desembolso.**

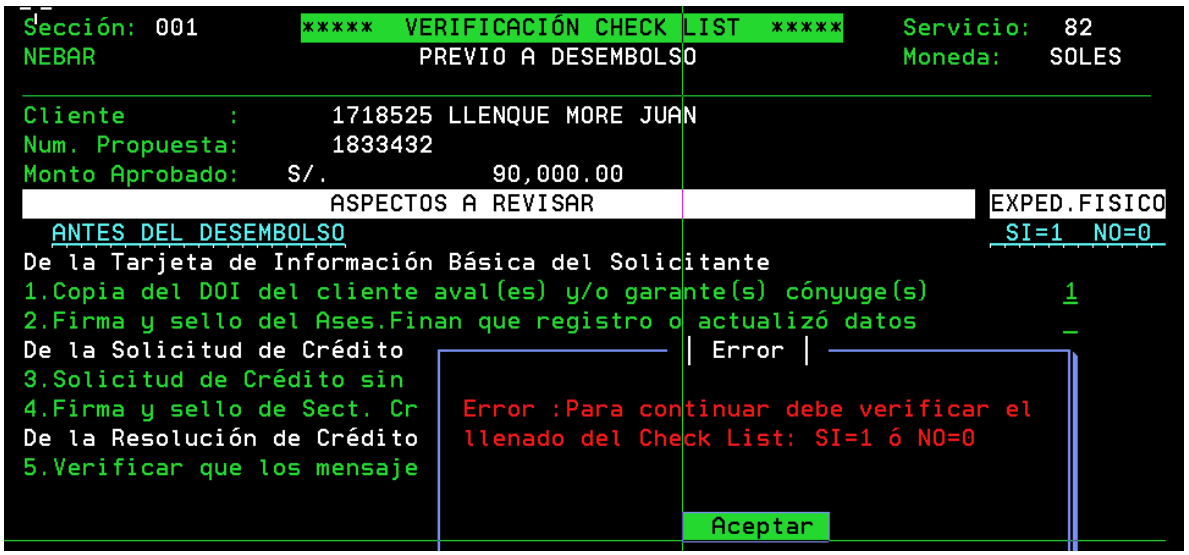


2) **El sistema nos muestra la lista de desembolsos que tiene dicha propuesta del cliente, aparece una nueva columna (CHECK P – Check List Previo), en primera instancia aparece "NO"**

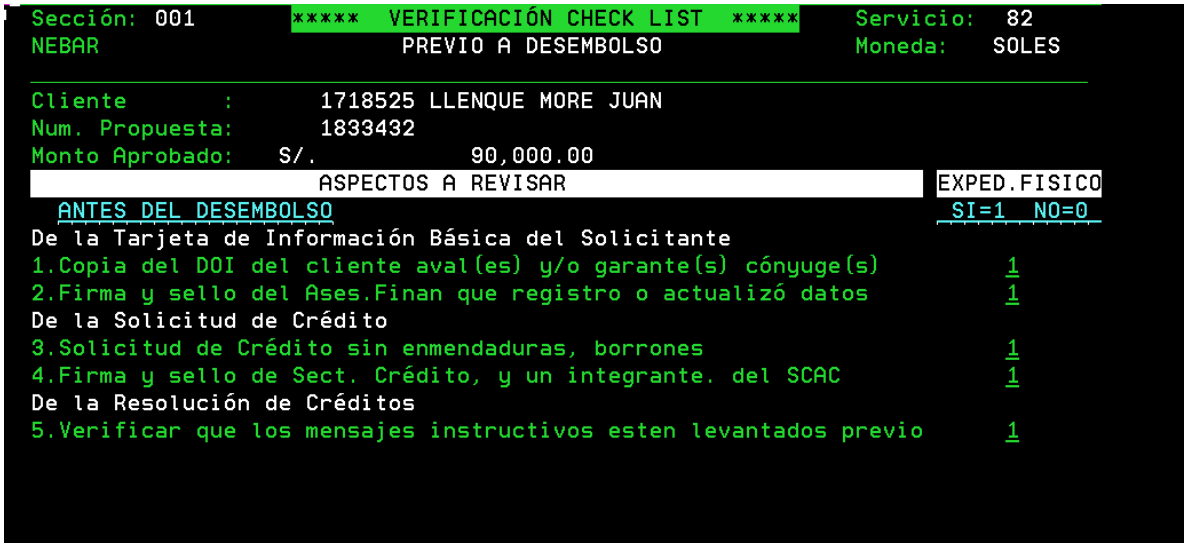


3) Al seleccionar el desembolso nos mostrará un expediente de check list previo al desembolso para ser llenado que aplica para personas naturales en este caso. (verificación de documentación física).

4) Debido a la cantidad de documentos del expediente a verificar se puede guardar el Expediente CL Previo como pendiente (Presionamos F6) para que sea llenado posteriormente, la columna de CL P se mantiene con NO.

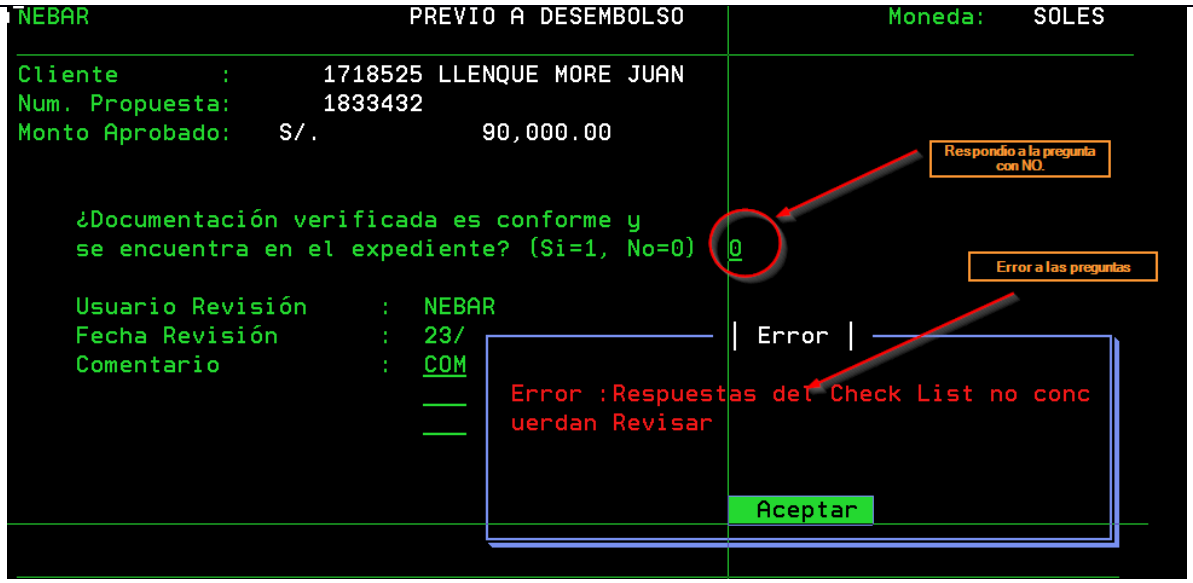


5) Si presionar ENTER para continuar con el llenado este nos mostrará un mensaje indicando que se debe verificar el llenado de Check List (todos deben marcarse).



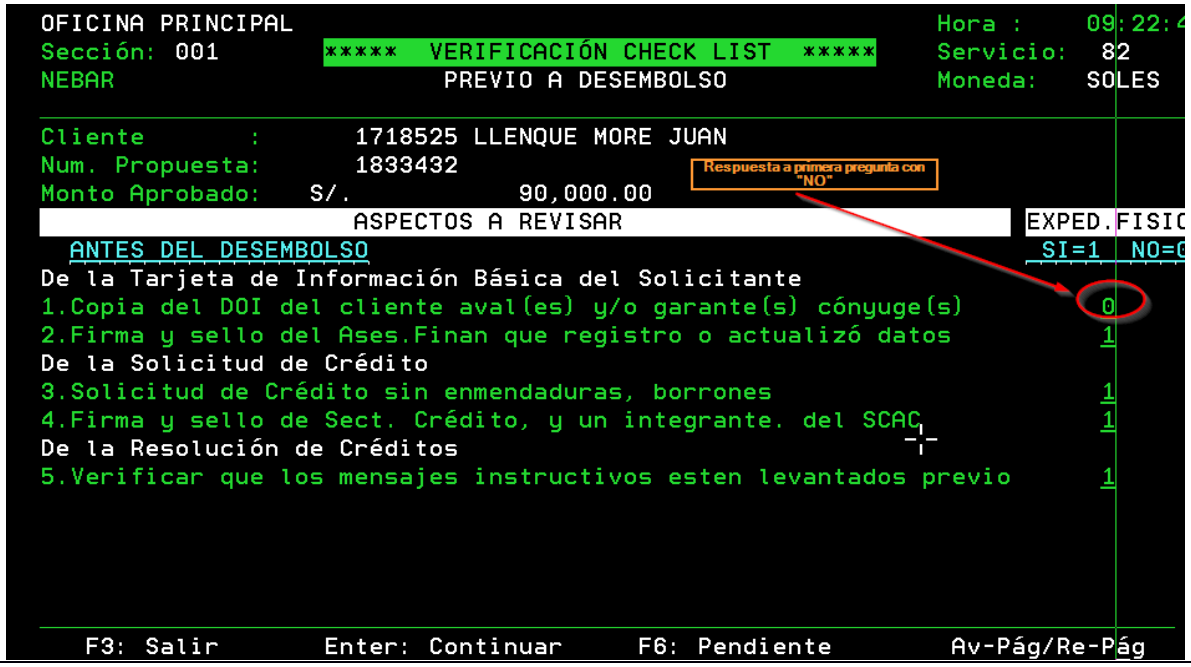
Presionamos la tecla "ENTER" para continuar con la siguiente pantalla.

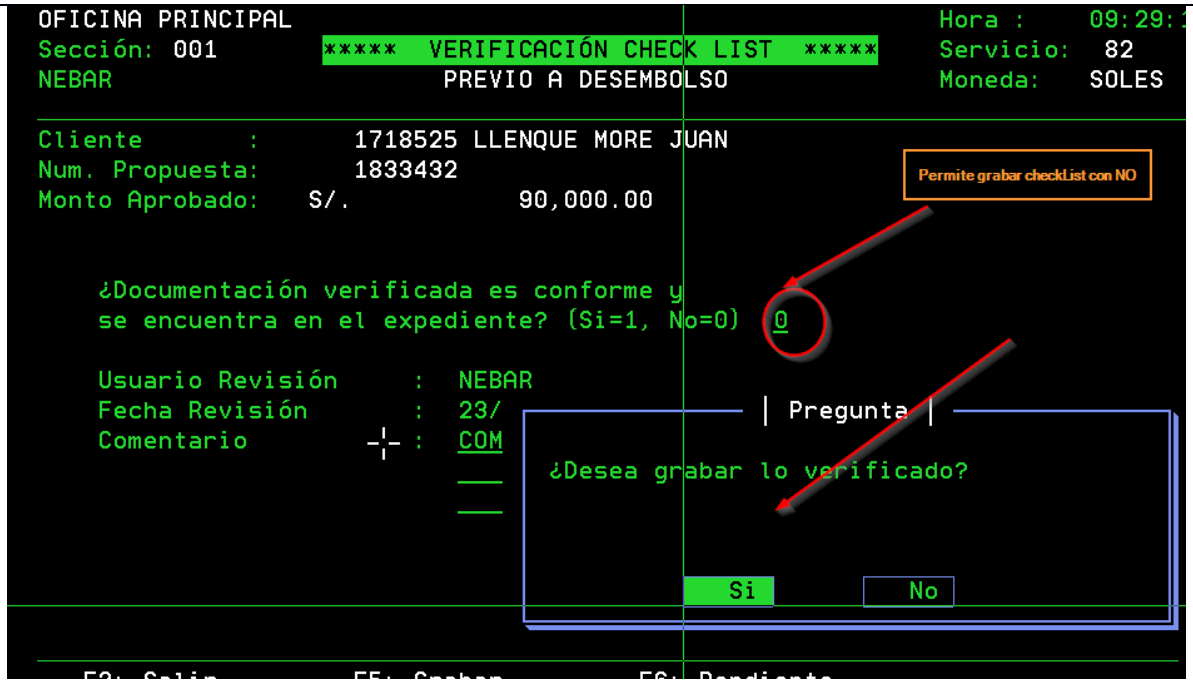
En el caso todas las preguntas del checkList están respondidas y la respuesta a la pregunta de la segunda pantalla es "SI", se dice que están alinedas, por lo tanto debe permitir guardar el checkList.

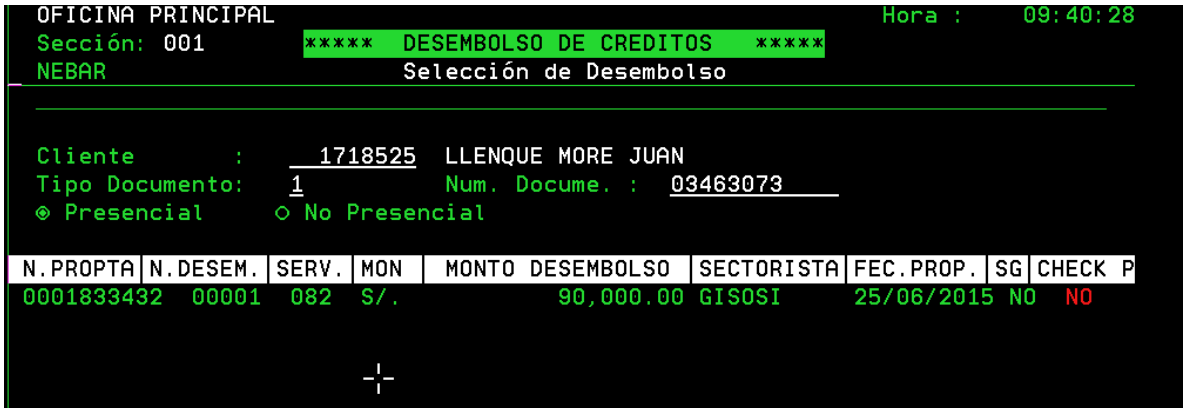


Si las preguntas están alineadas y se responde a la pregunta con NO el sistema muestra en error que se muestra en la pantalla anterior, quiere que decir que las preguntas no se encuentran alinedas.

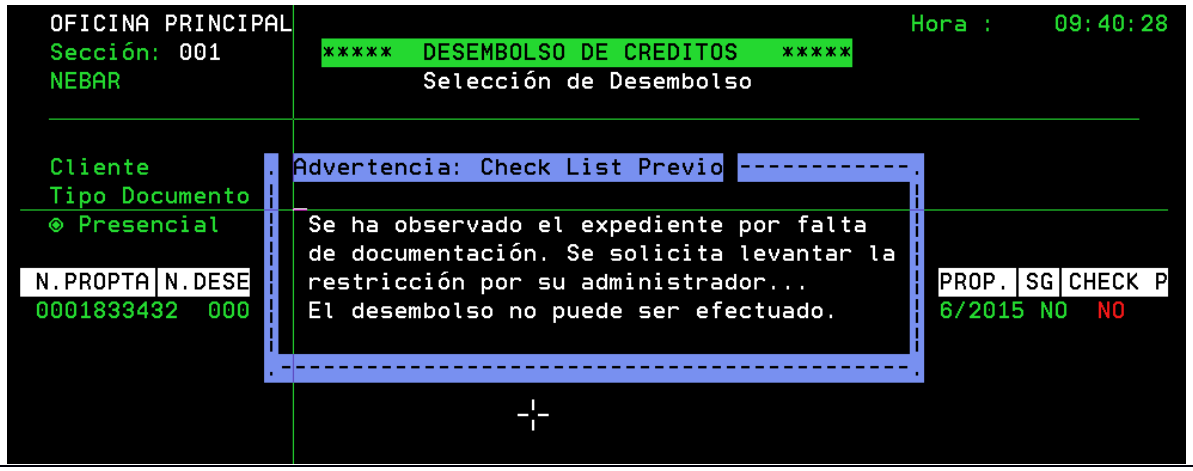
6) Caso en que un Checklist Previo al desembolso se puede guardar con Respuesta "NO" es el siguiente caso, esto aplica tanto a personas natural como a persona jurídica:





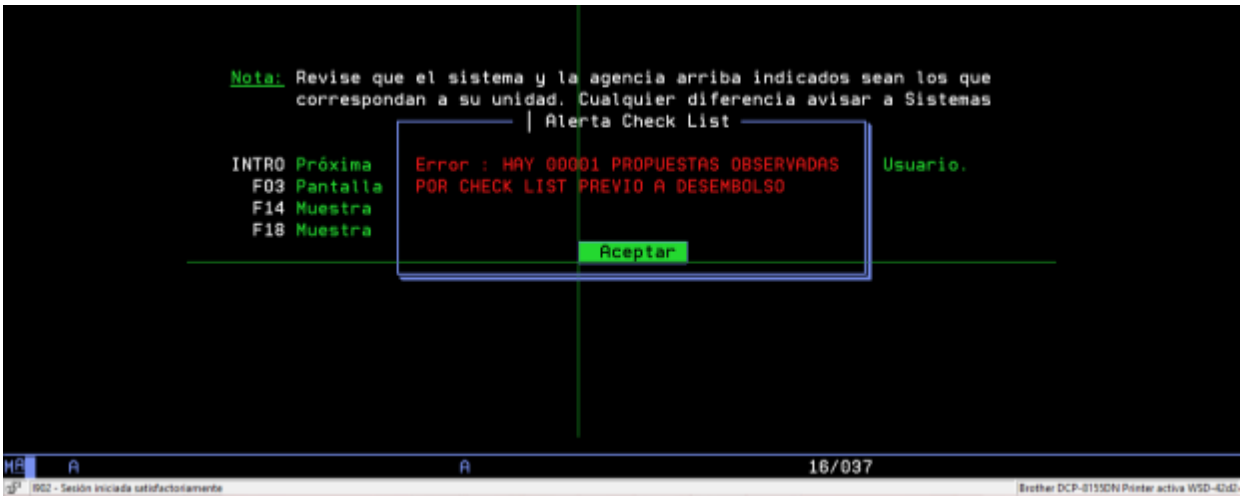


7) Validamos la pantalla según corresponda, si es SI=1 (el sistema validará correctamente el llenado de check list para luego pasar al siguiente proceso). Ir al paso 21.



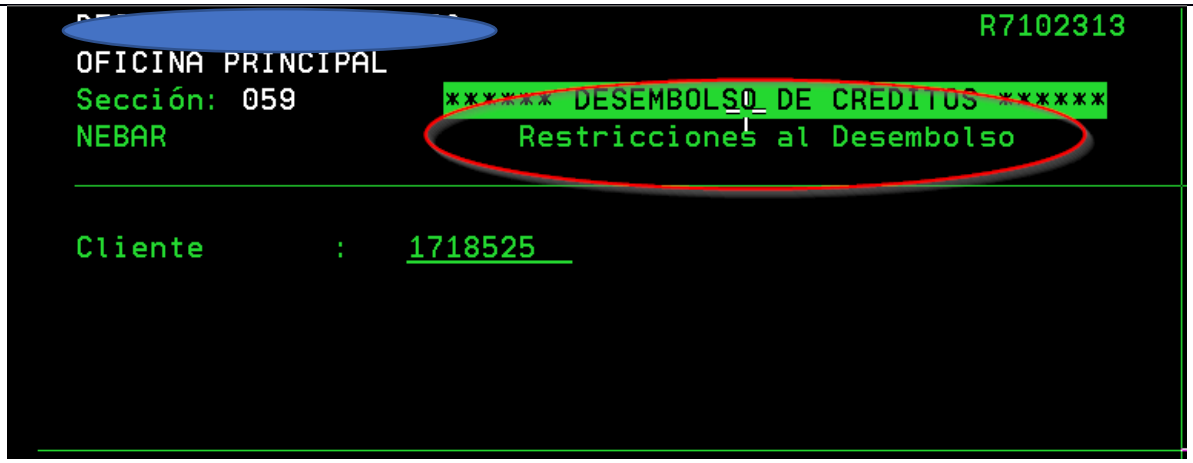
8) Si volvemos a seleccionar el desembolso el sistema me lanza un mensaje indicándome que se tiene que levantar la restricción por el administrador.

MENÚ ADMINISTRADOR(Visualización de Alerta y levantamiento de restricción)

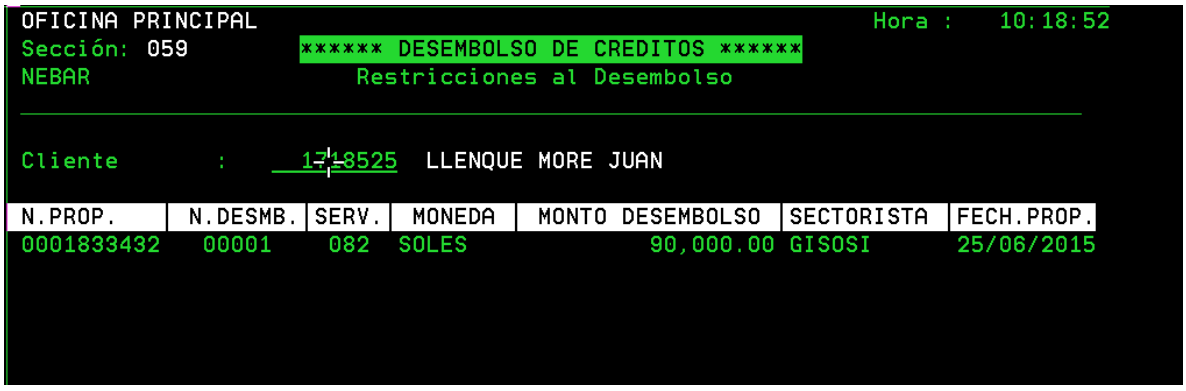


9) Cuando el administrador ingrese al SOFIA a través de su usuario y contraseña, este le aparecerá un mensaje indicando la cantidad de propuestas observadas por Check List previo al desembolso, en caso no tuviera no se muestra el mensaje.

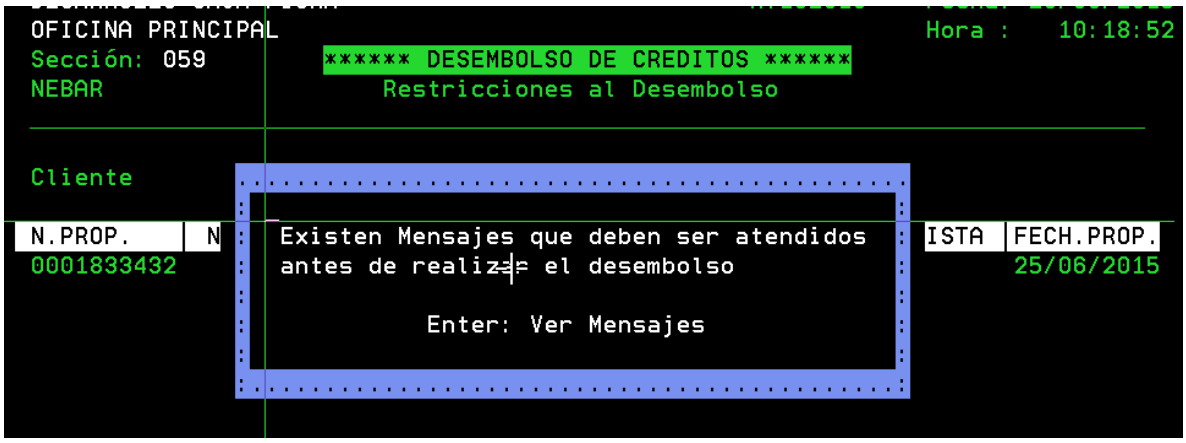
10) Para levantar las observaciones el administrador tendrá que ingresar al menú de levantamiento de restricciones (59)



11) Ingresa el código de cliente. Ejm: 1718525

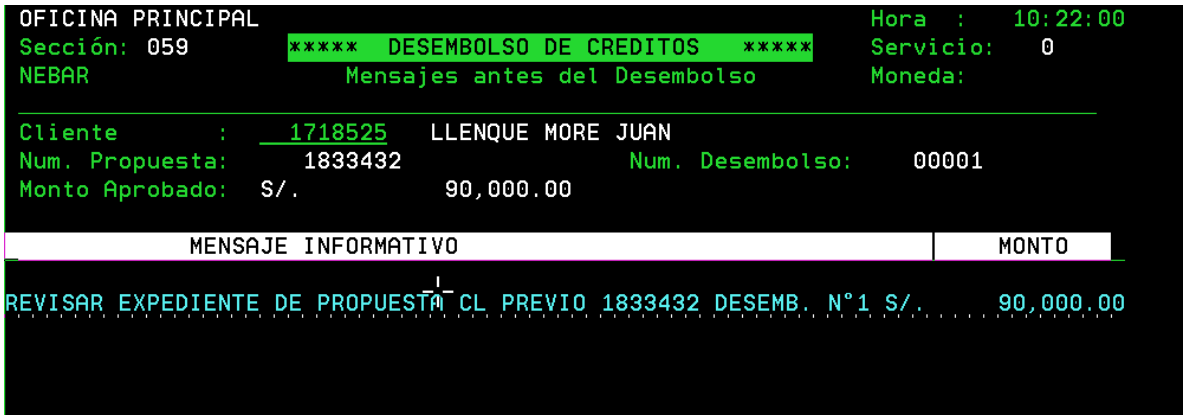


12) Muestra el desembolso al cual se le ha aplicado la restricción.

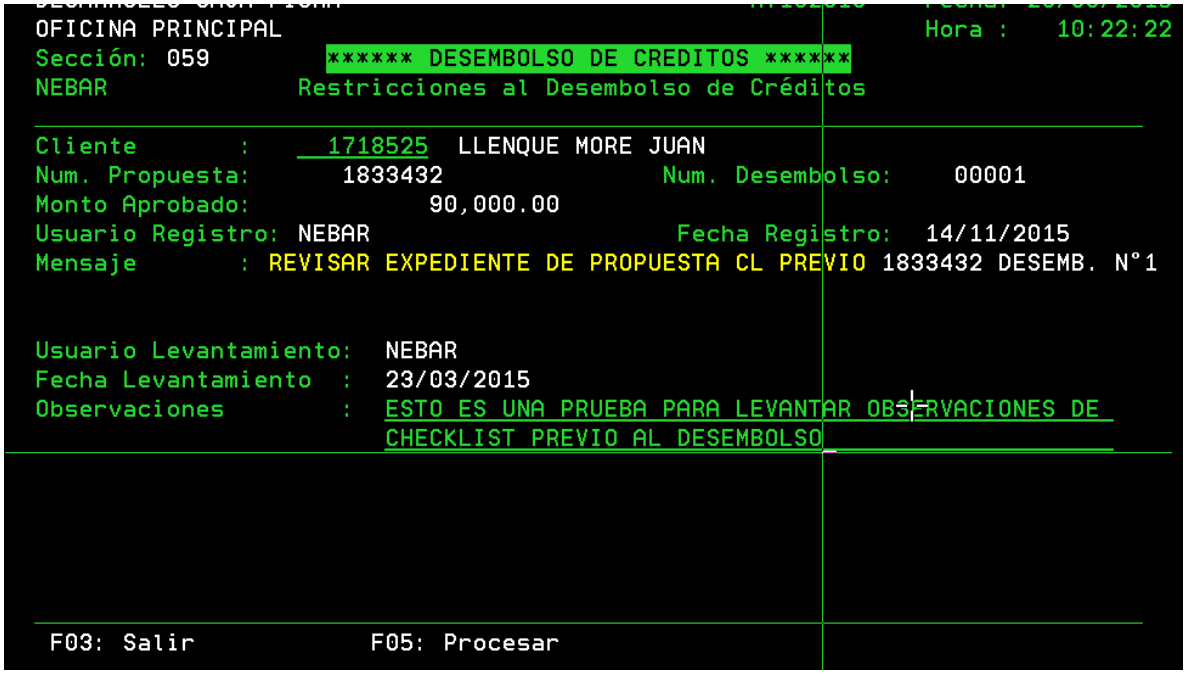


13) Al seleccionar la propuesta, el sistema me indica que: Existen mensajes que deben ser

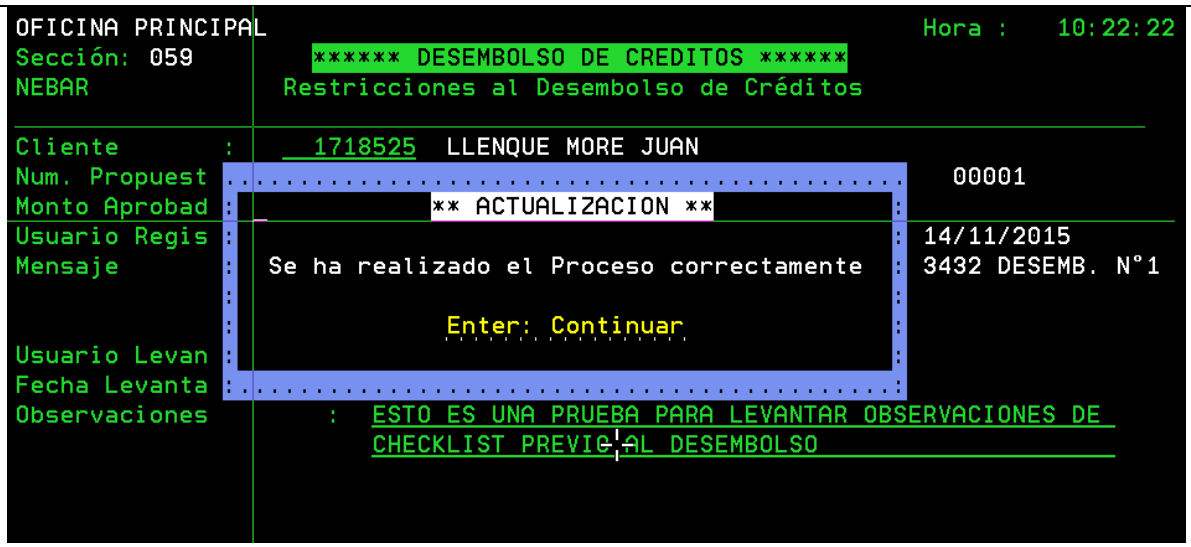
atendidos antes de realizar desembolso.



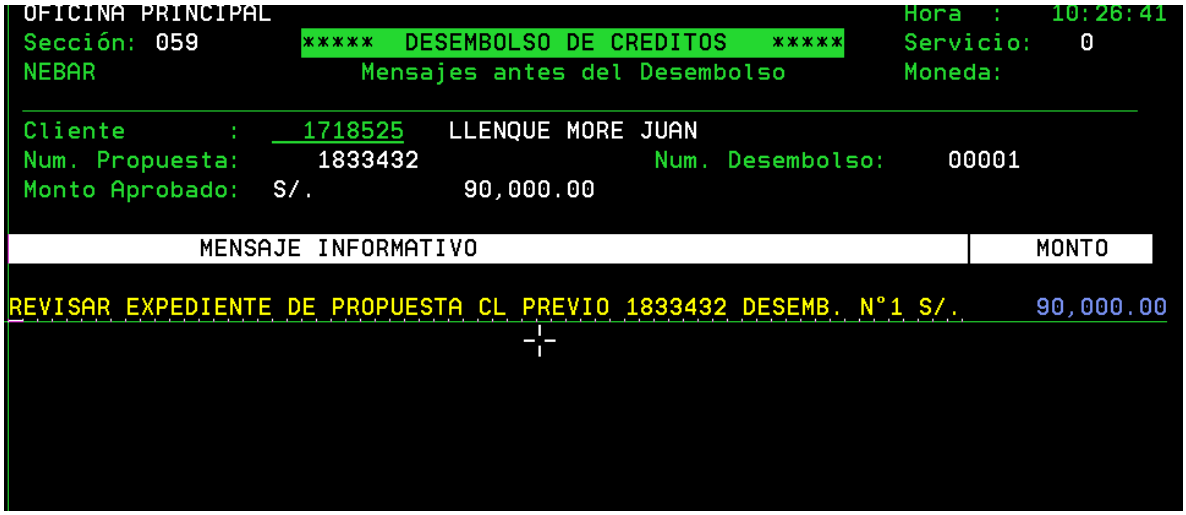
14) Muestra la pantalla indicando que hay que revisar el expediente check list previo, el cual ha sido observado, para ello seleccionamos para levantar la restricción.



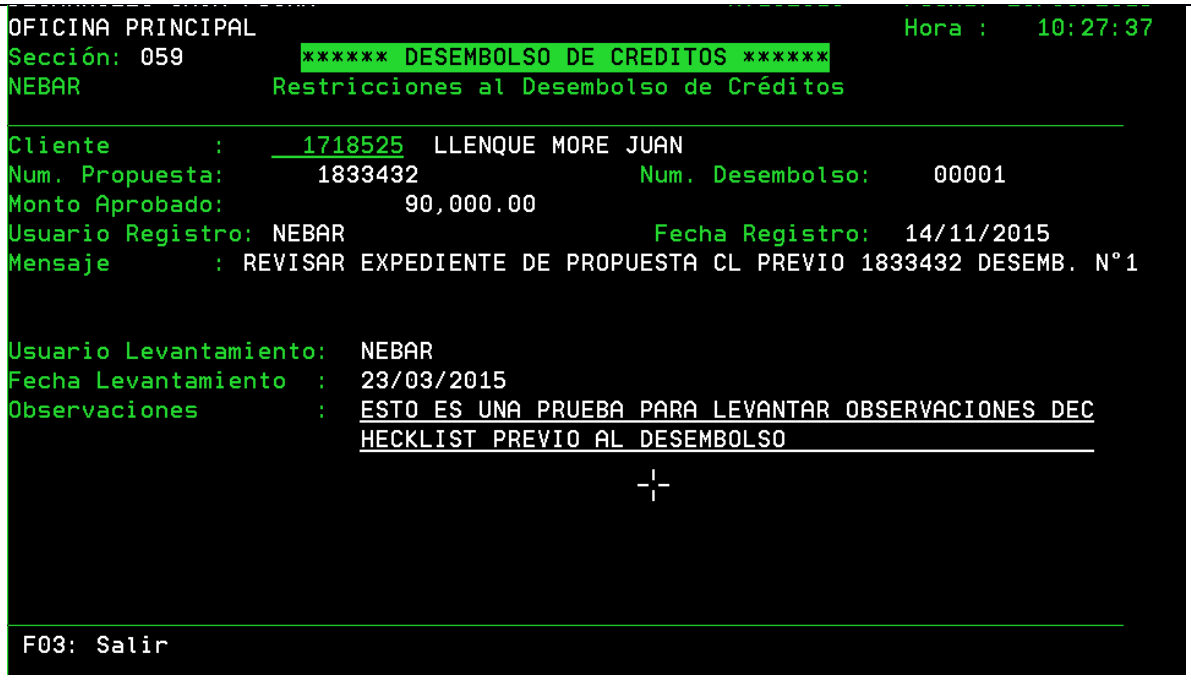
15) El administrador emite su comentario para levantar la restricción



16) Presionamos F05 para procesar la restricción de check list observada.

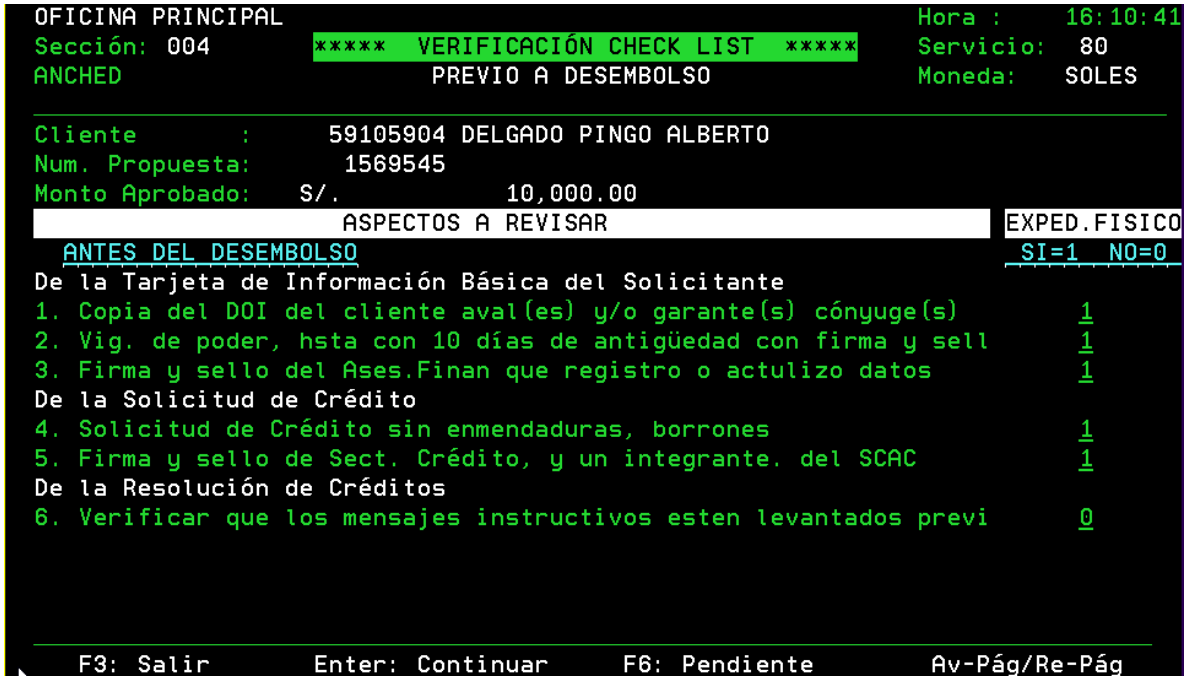


17) En la pantalla se muestra el cambio de color amarillo del mensaje que me indica que ha sido levantada la observación.

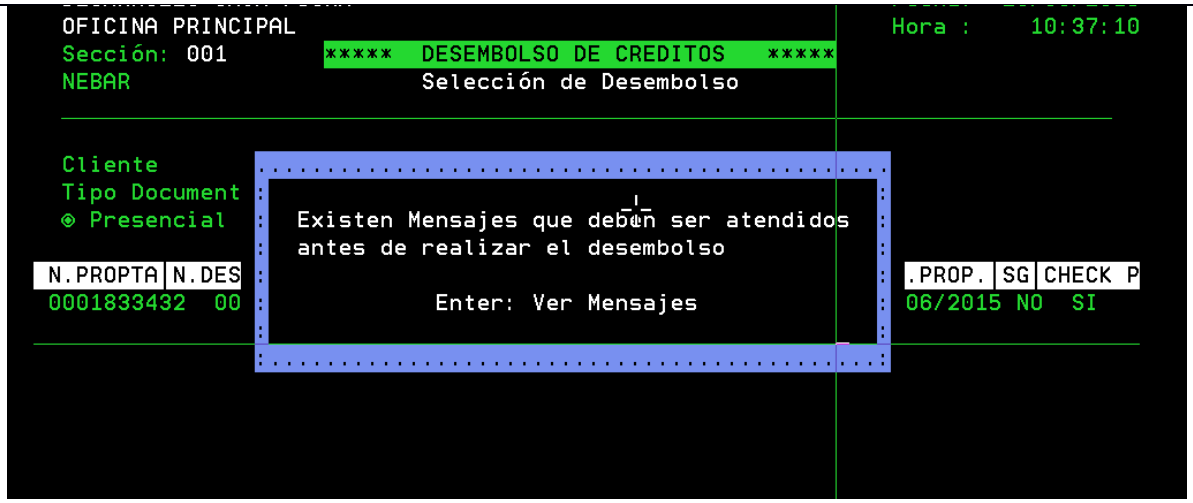


18) Aquí visualizamos el mensaje de la restricción de la propuesta antes del desembolso.

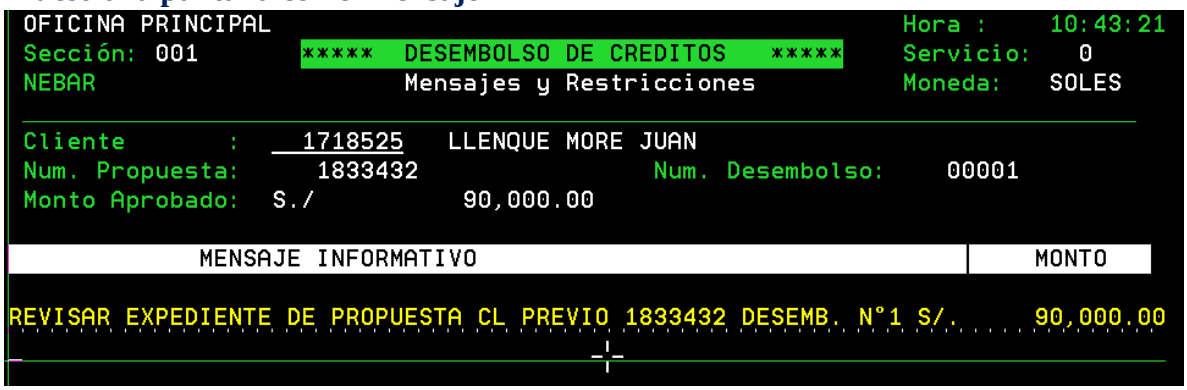
MENÚ DE OPERACIONES (APROBAR EXPEDIENTE)



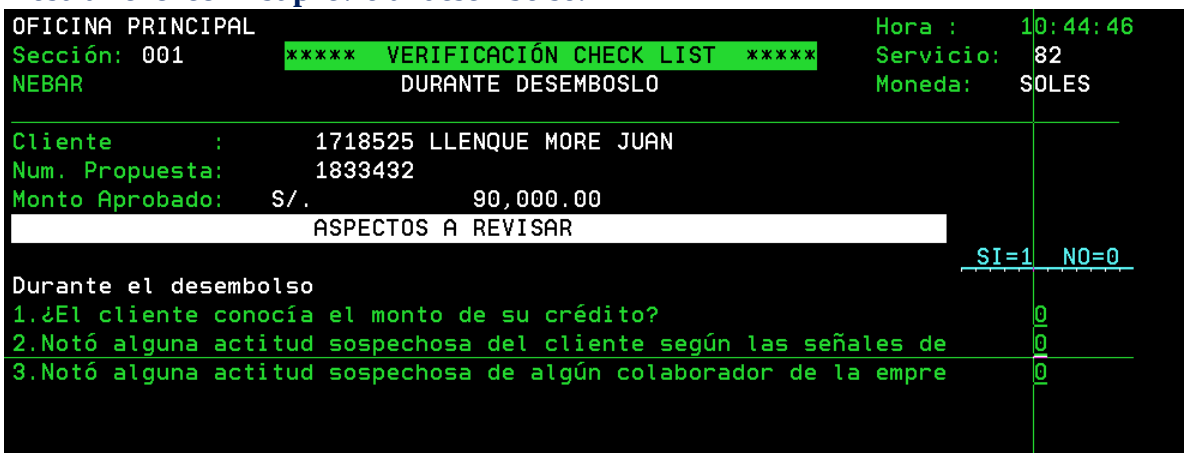
19) Una vez levantada la restricción de la propuesta ingresamos nuevamente al desembolso, esta vez ya no debe mostrarnos el checkList Previo, por el contrario si se ha seleccionado Desembolso Personal debe pasar al CheckList Durante el desembolso.



Muestra la pantalla con el mensaje



Luego presionamos la tecla F05 para pasar al siguiente paso, en este paso ya no debe mostrar el checkList previo al desembolso.

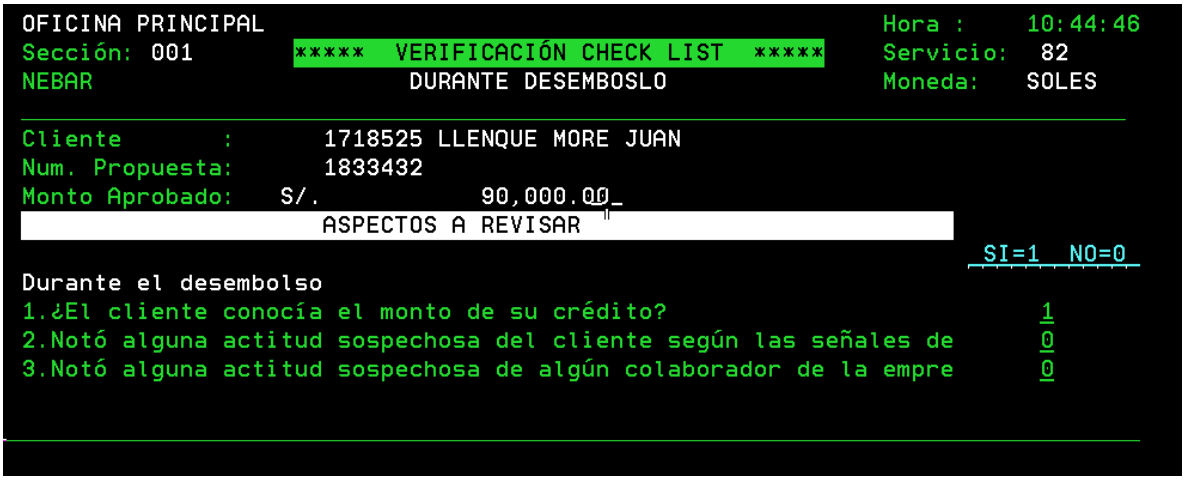


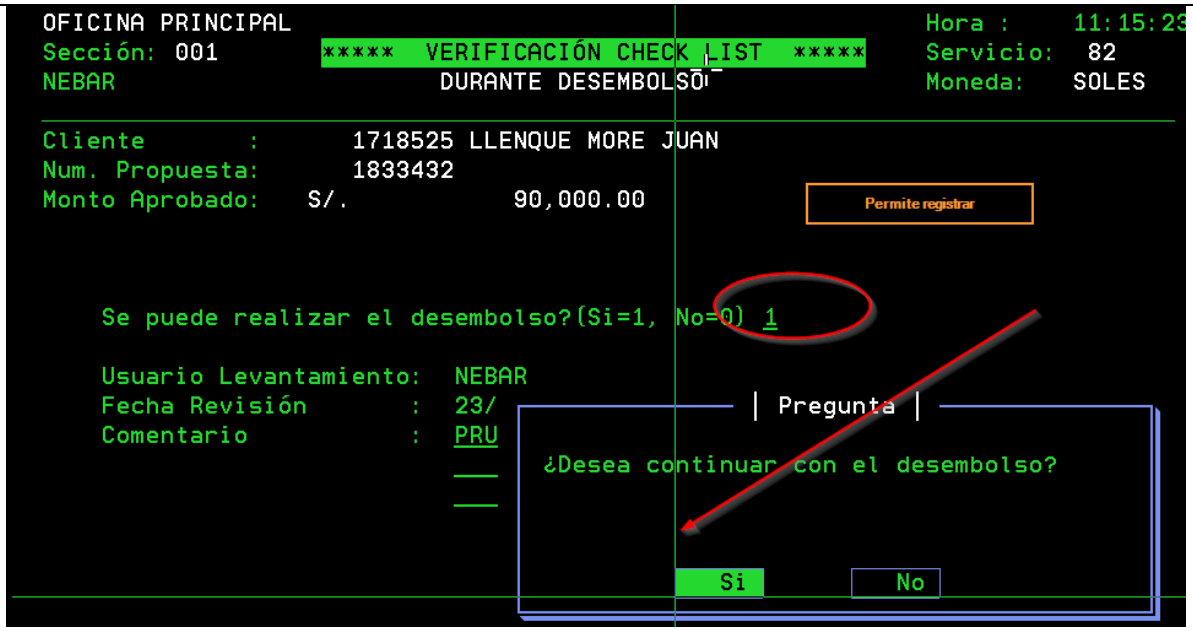
Como se puede ver, al procesar va directo al checklist durante el desembolso.

20) Presionamos ENTER para continuar y visualizamos el último comentario de la revisión, por consiguiente se da por aprobada dicho check list.

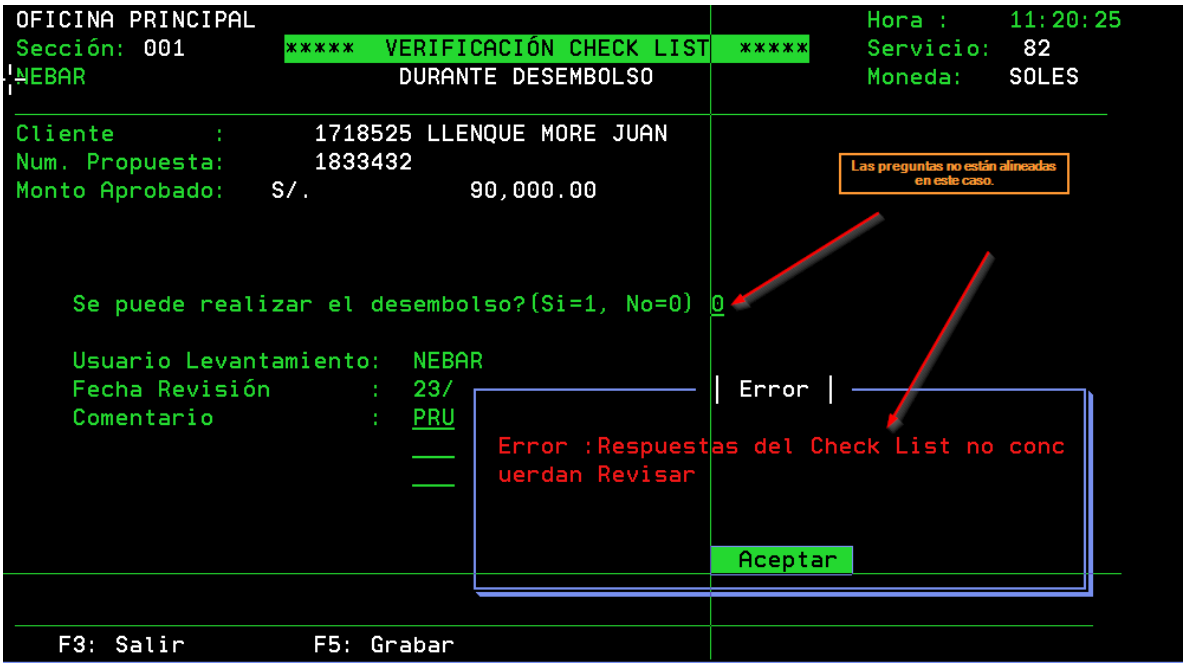
21) Presionamos F5: Grabar para dar pase al siguiente proceso que muestra un mensaje de confirmación (SI/NO).

En este paso, es importante que para proceder con el desembolso las respuestas tiene que estar alineadas, es decir que la primera pregunta se responda con si y las restantes se respondan con y la respuesta a la pregunta sea SI, caso contrario el sistema no debería permitir continuar con el siguiente paso:

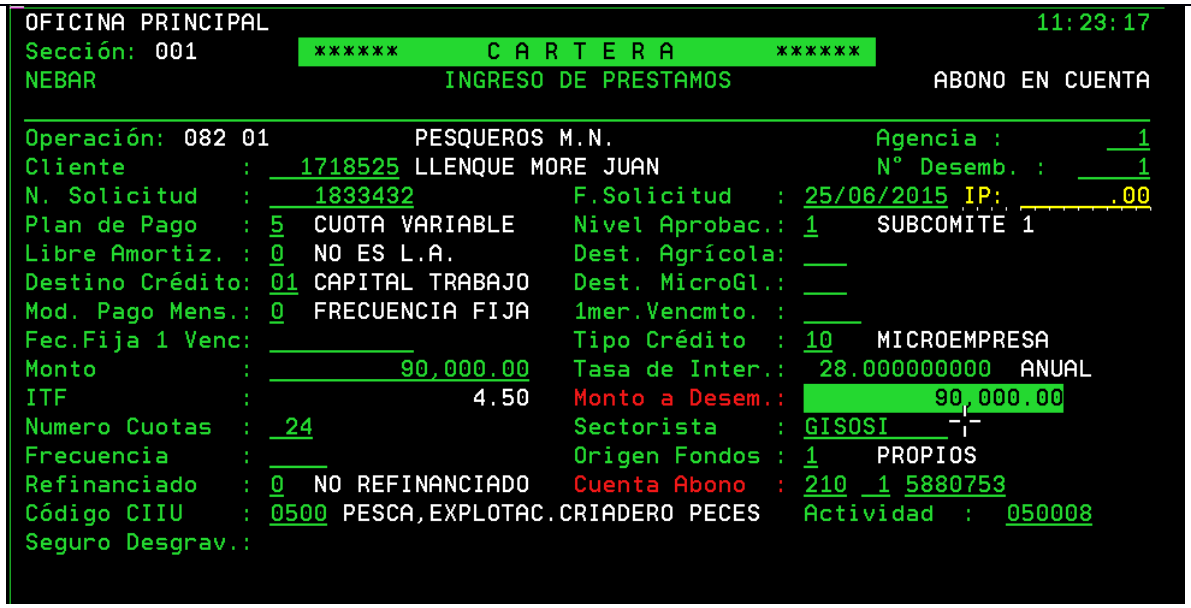




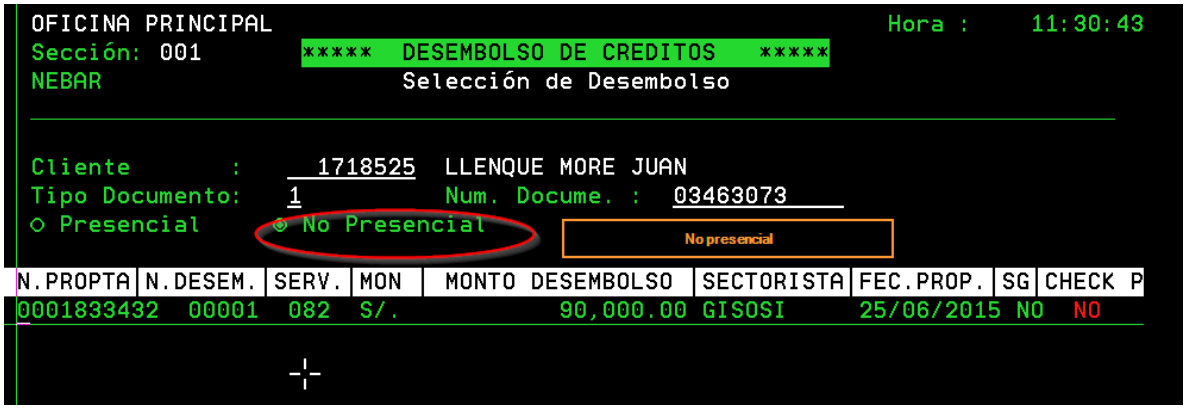
En caso no se cumpla estas condiciones, no debe permitir registrar.



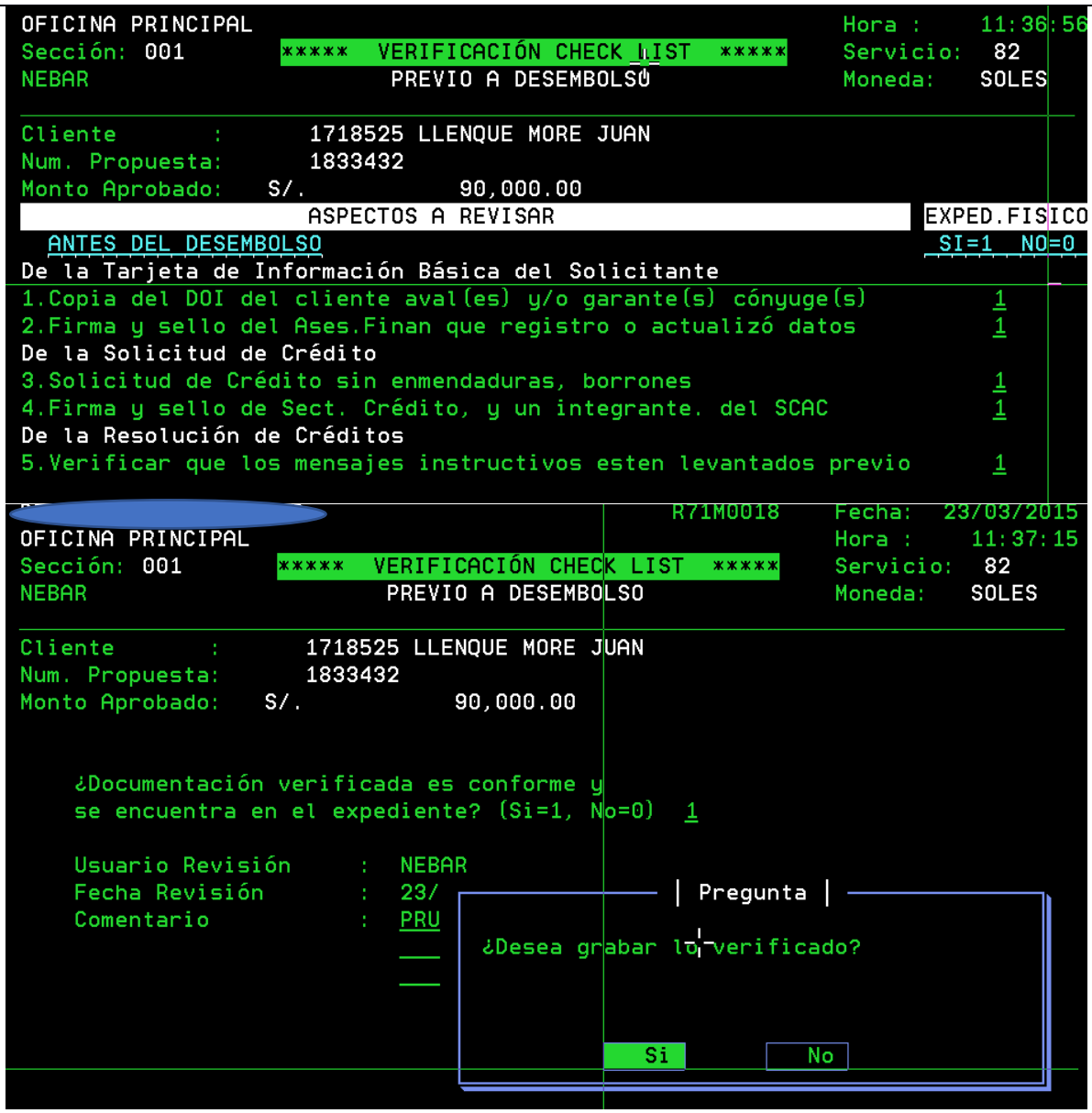
Si todo ha seguido el flujo normal se mostrará la pantalla del desembolso.



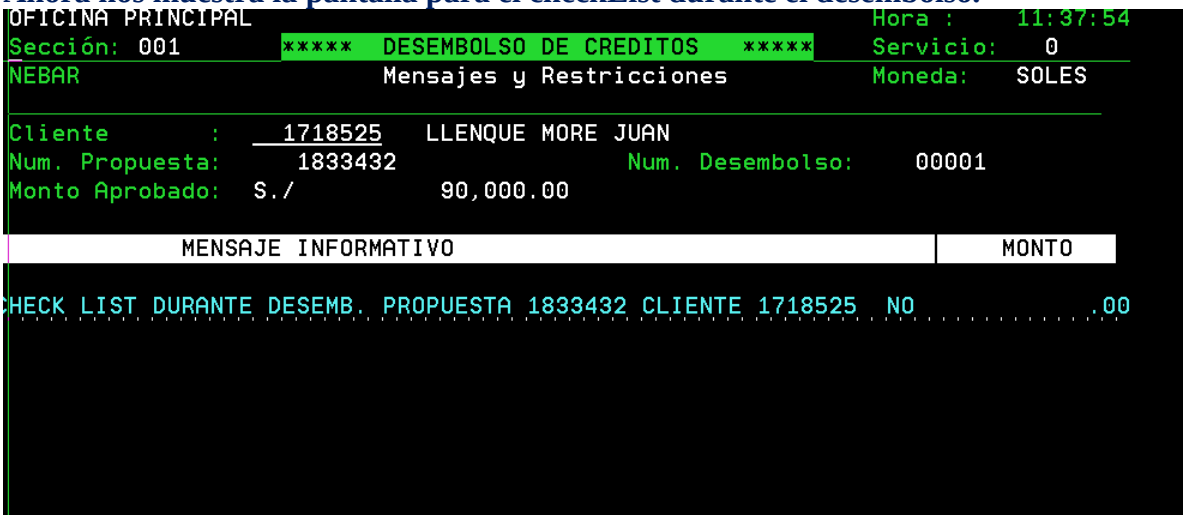
22) En cuanto a lo relacionado a la opción de CheckList que no sea presencial , el sistema no debe mostrar la pantalla para el check list durante el desembolso.



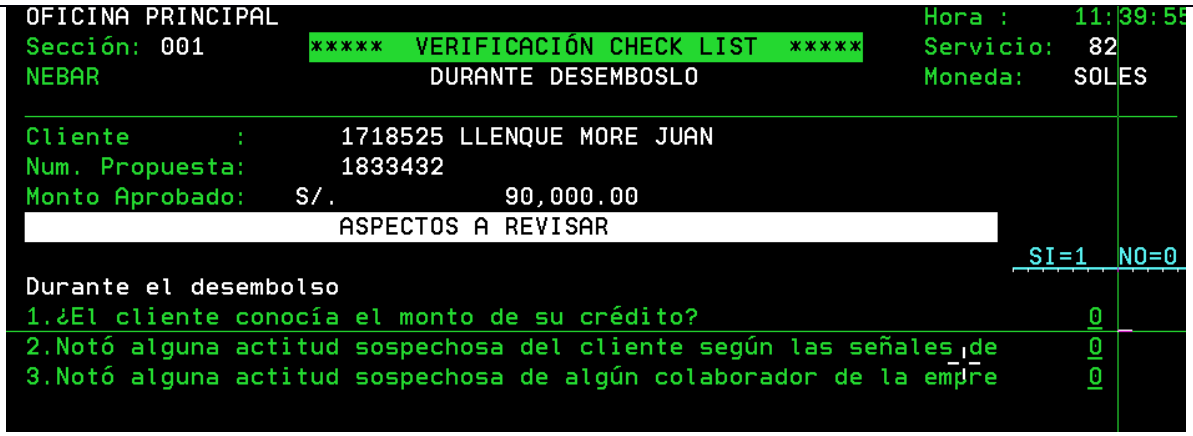
Ahora respondemos de forma tal que las repuesta estén alineadas y permita pasar directamente a la pantalla de desembolso.



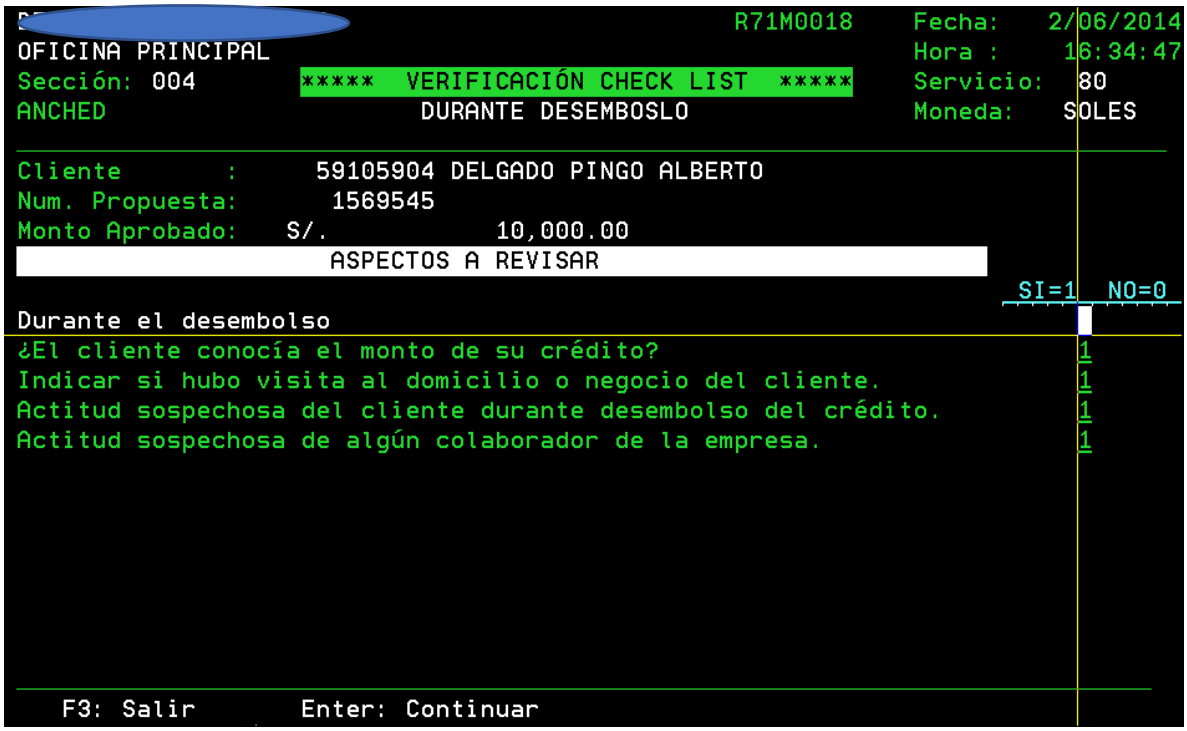
Ahora nos muestra la pantalla para el checkList durante el desembolso.



Al presionar la tecla F5 pasará directamente a la pantalla del desembolso.

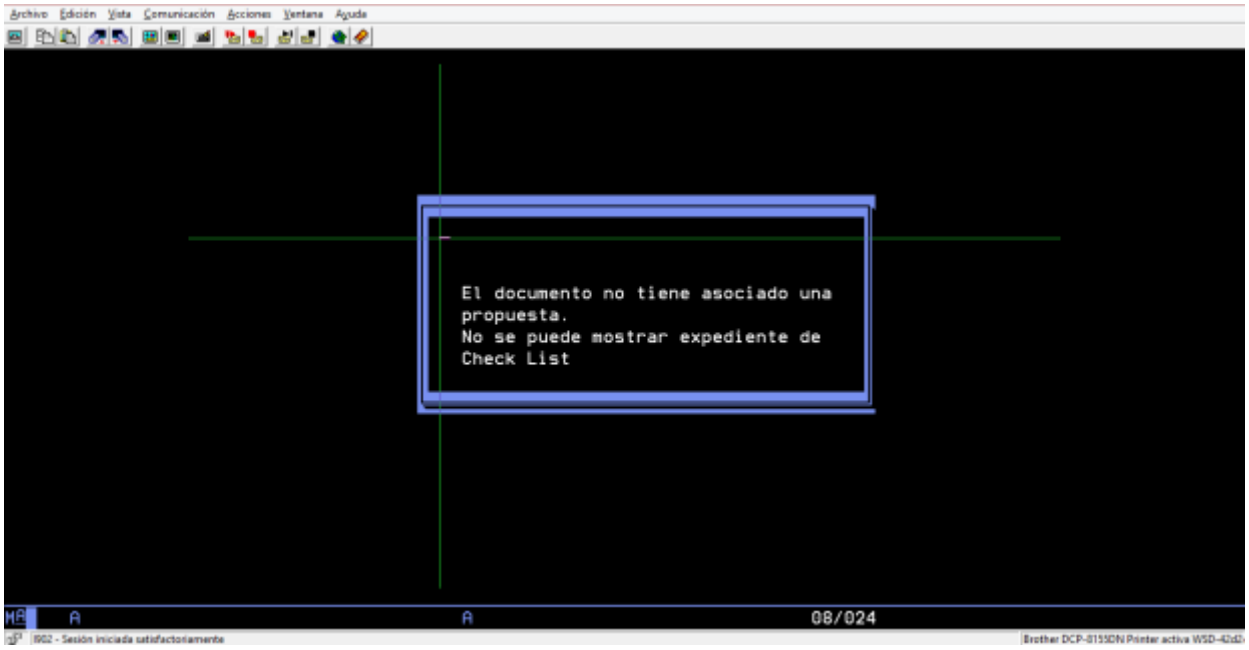


23) Si se guarda como pendiente y visualizamos los mensajes que deben ser atendidos, en la parte de CHECK LIST DURANTE DESEMBOLSO DE PROPUESTA sigue como NO.



TRANSACCIÓN 3: CONSULTA DE DOCUMENTOS

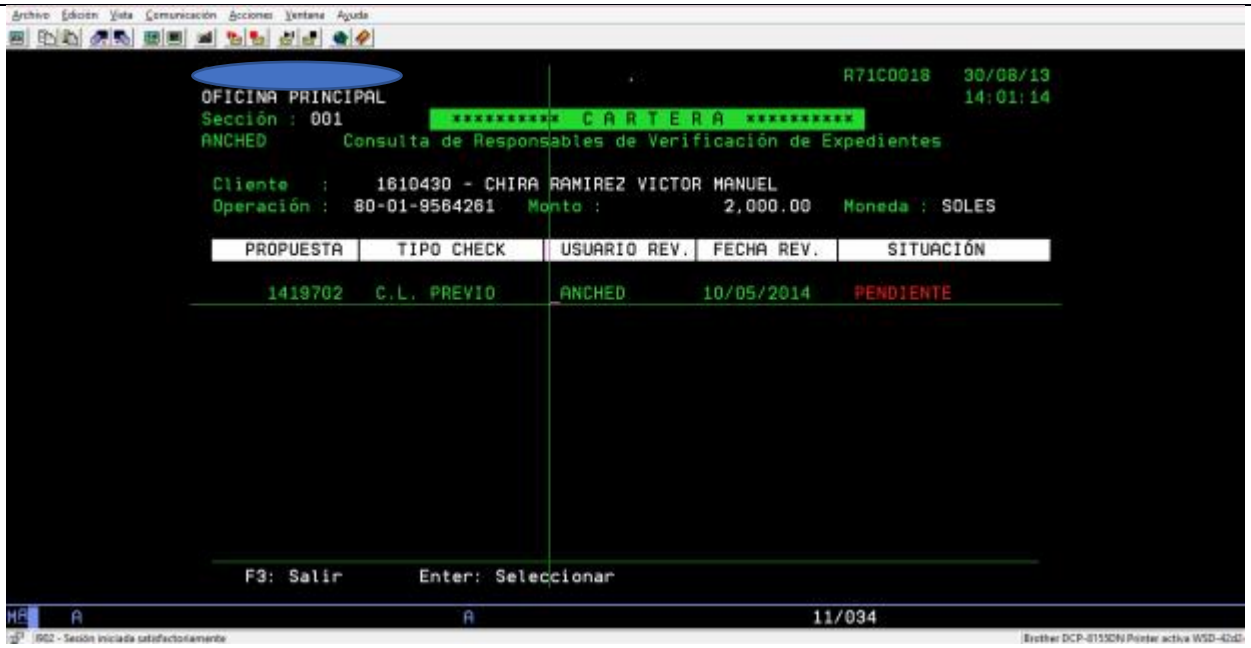
24) En la consulta de documentos aparece un nuevo menú F15: Chk.List donde visualizaremos quién ha realizado el Check List tanto previa como durante el desembolso.



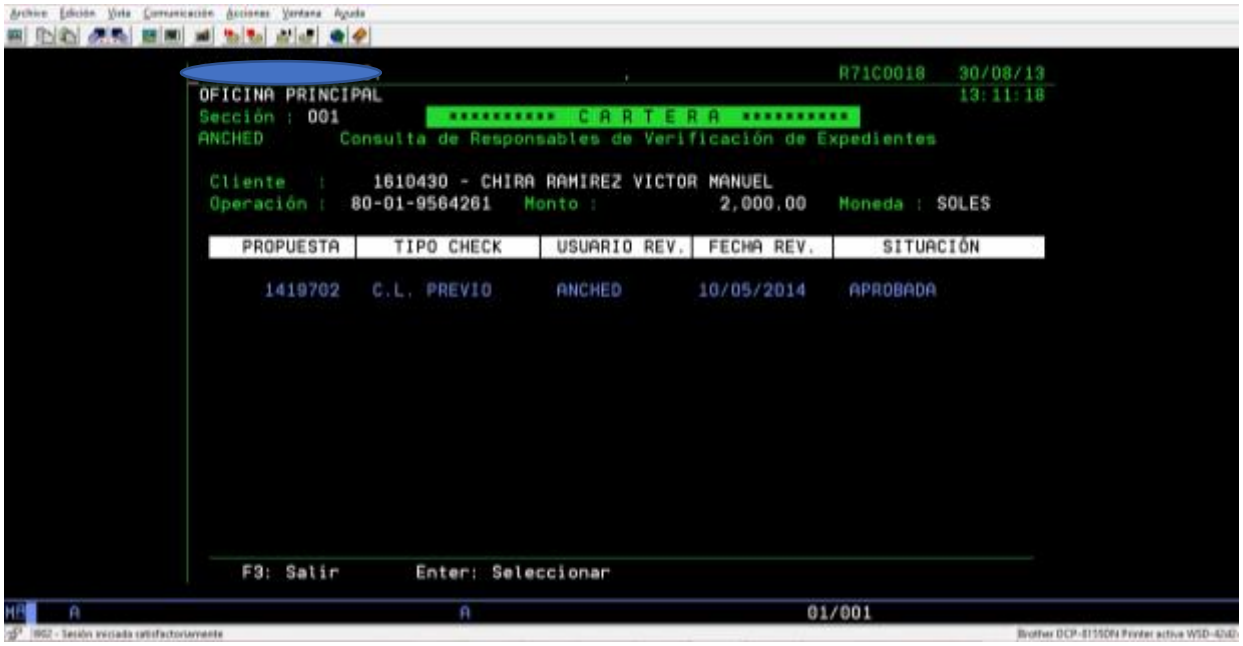
25) Si el documento no tiene asociada una propuesta nos muestra la siguiente pantalla.

26) Se visualiza la siguiente pantalla, donde no se muestra ningún movimiento de check list.

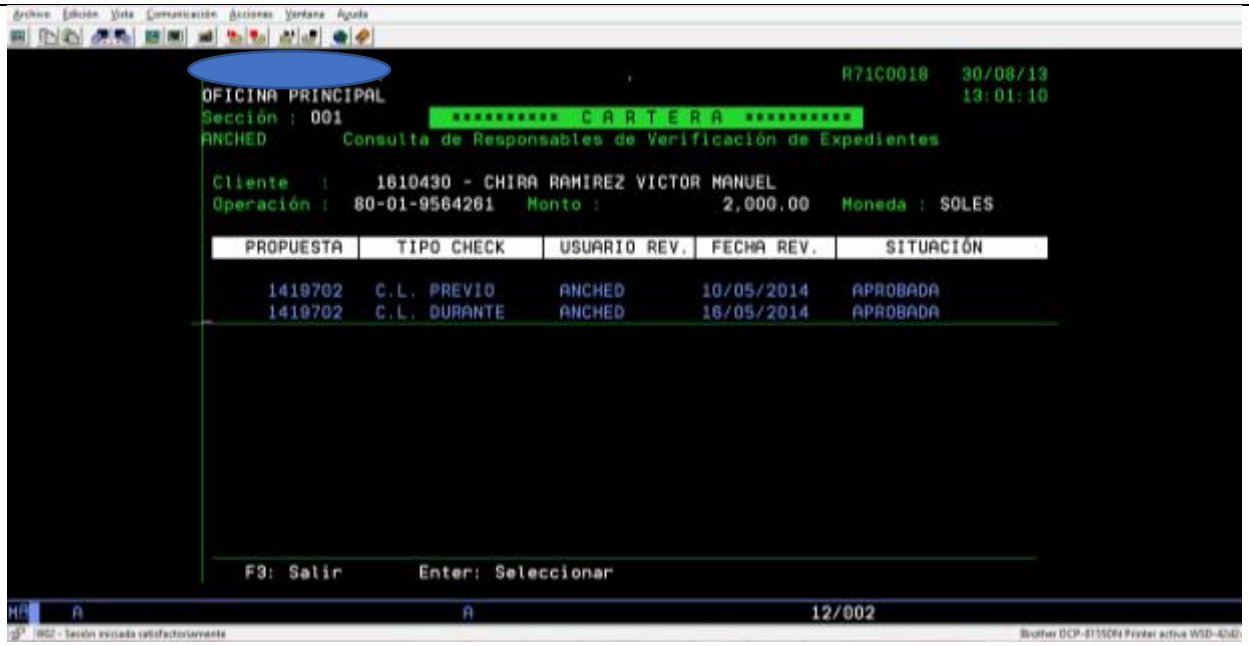
27) Si elegimos un documento con propuesta que tenga movimientos de quien ha realizado el check list.



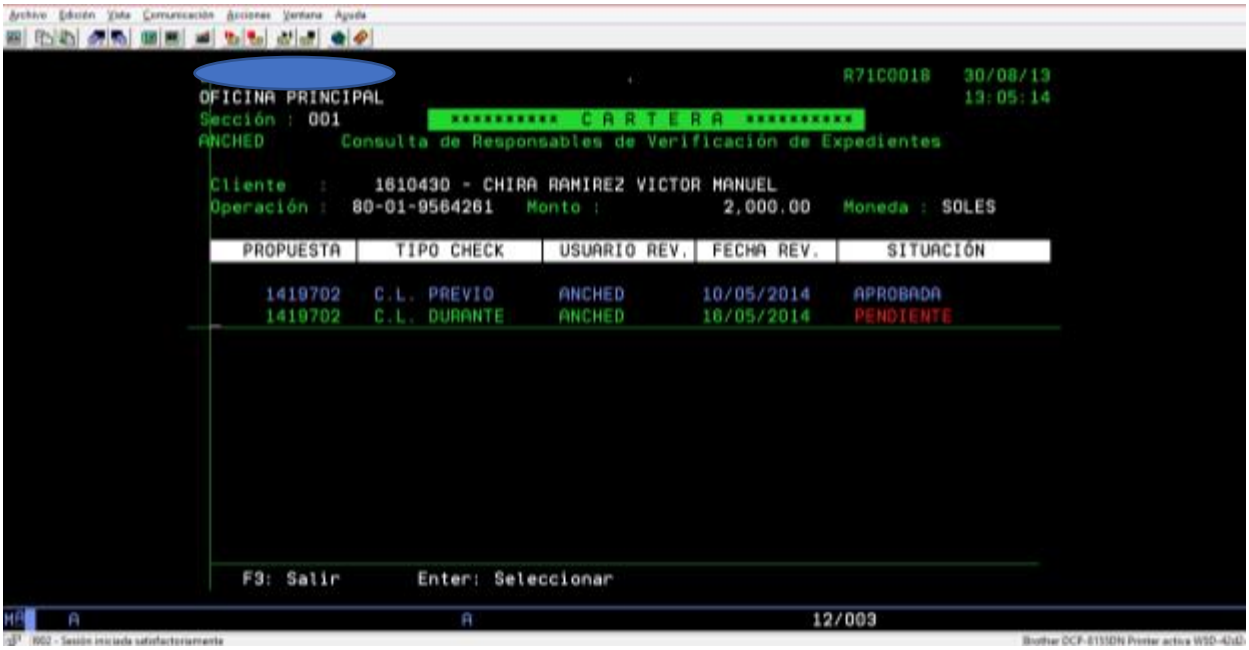
28) No muestra la siguiente pantalla con Situación "PENDIENTE", significa que aún no se ha terminado de llenar el check list ya sea por una observación o falta de documentación.



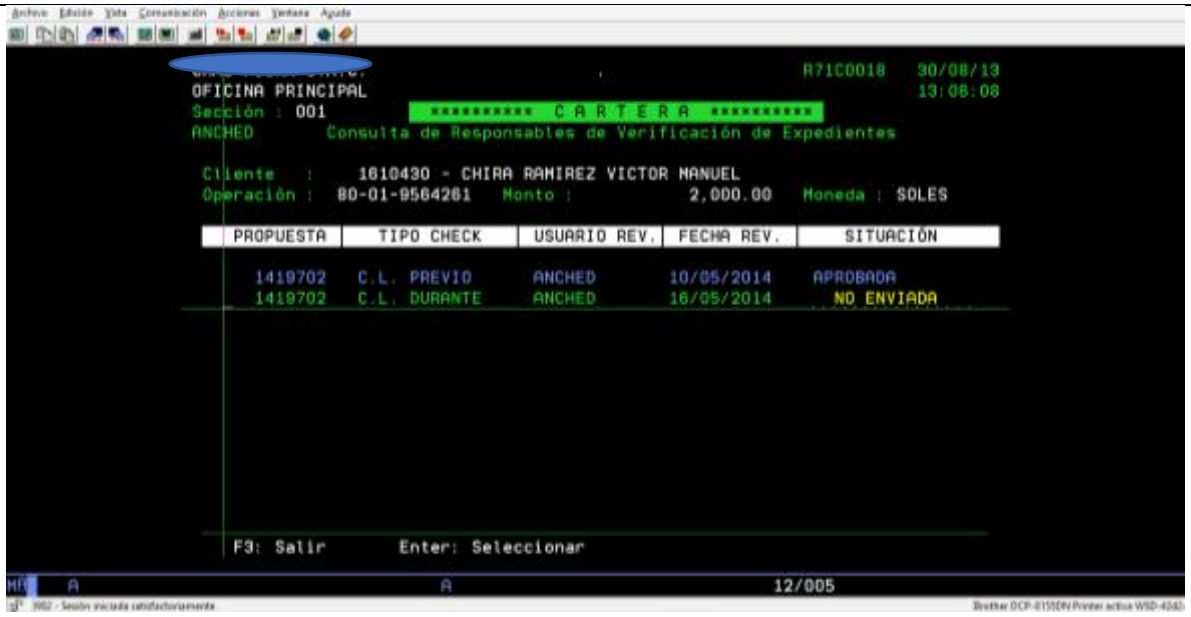
29) O nos muestra una pantalla con situación "APROBADA" que significa que el check list ha sido aprobado previo al desembolso.



30) O también nos muestra donde C.L. Previo como C.L. durante el desembolso han sido aprobadas correctamente, indica que ya se puede realizar el desembolso.

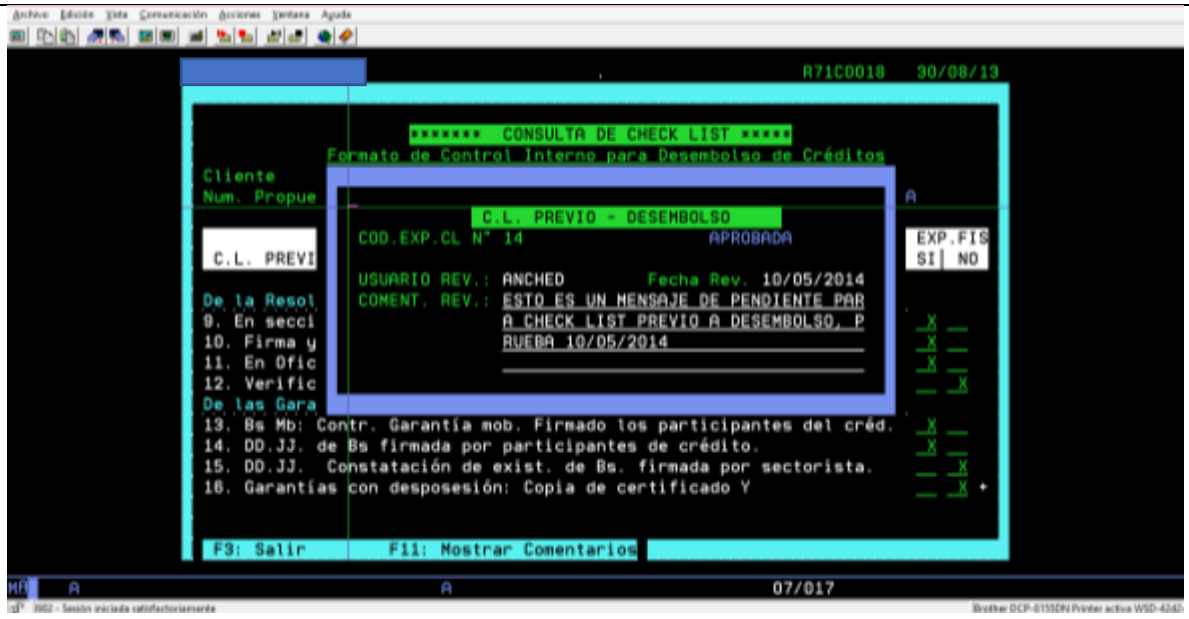


31) En este caso nos encontramos que el C.L durante esta pendiente de aprobación por el usuario ANCHED.



32) **En este caso nos muestra que el check list esta siendo revisado por el usuario ANCHED, si antes validar el check list (Aprueba o deja como pendiente de aprobación).**

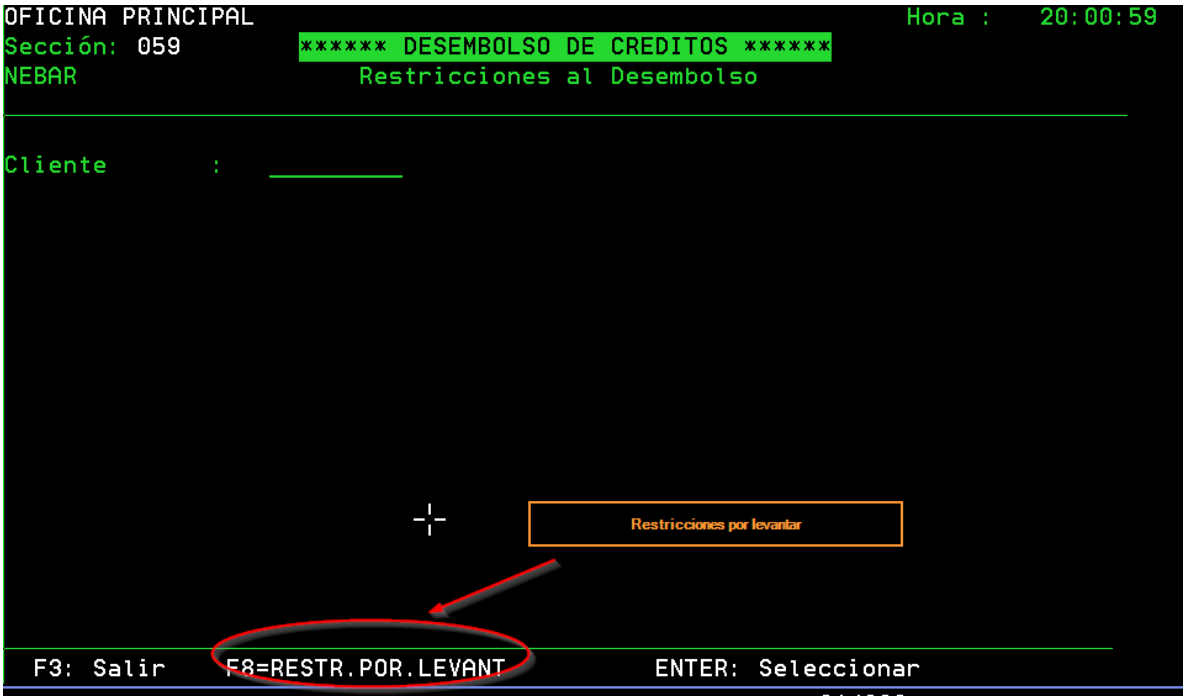
33) **Podemos visualizar el detalle de cada check list que ha sido llenado según sea el caso tanto previo como durante el desembolso.**



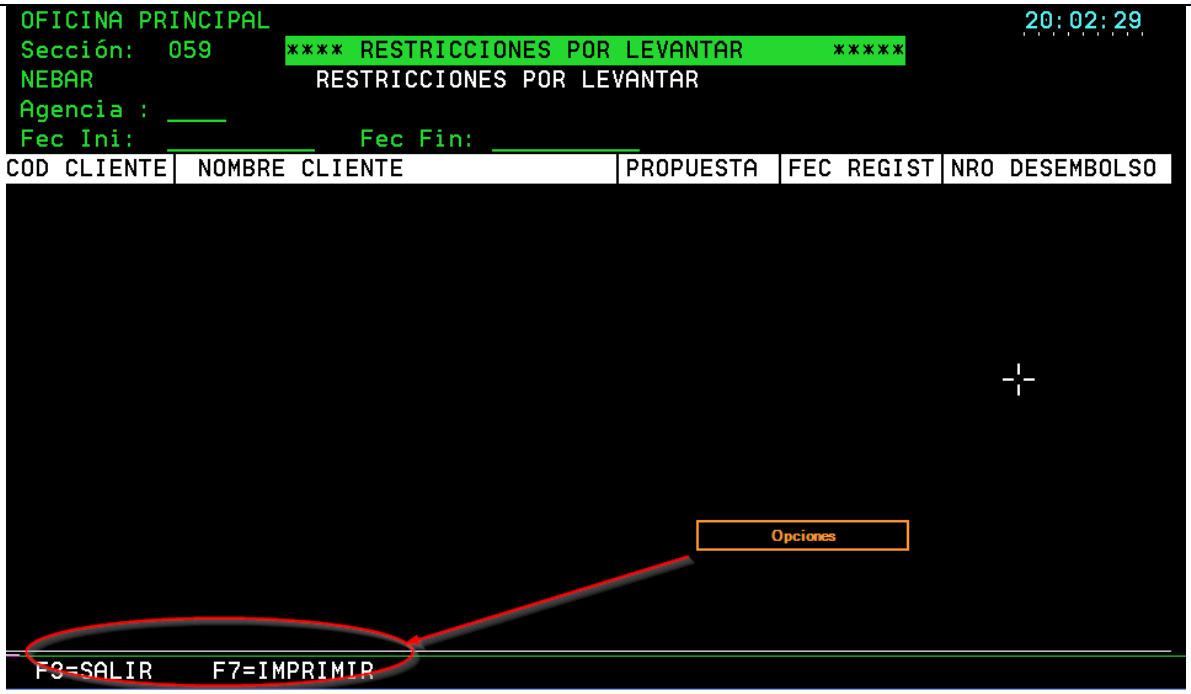
34) En esta pantalla podemos visualizar los comentarios que se han hecho previo o durante las revisiones de los check list.

REPORTES :

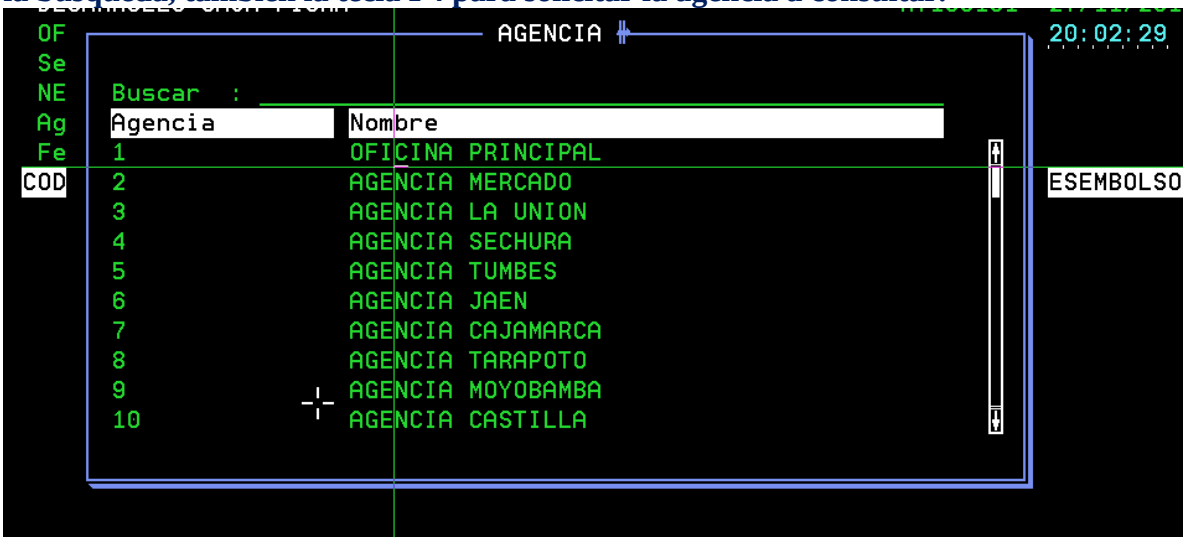
36) En la opción 59 Levantamiento de restricciones se implementó la opción para poder visualizar.



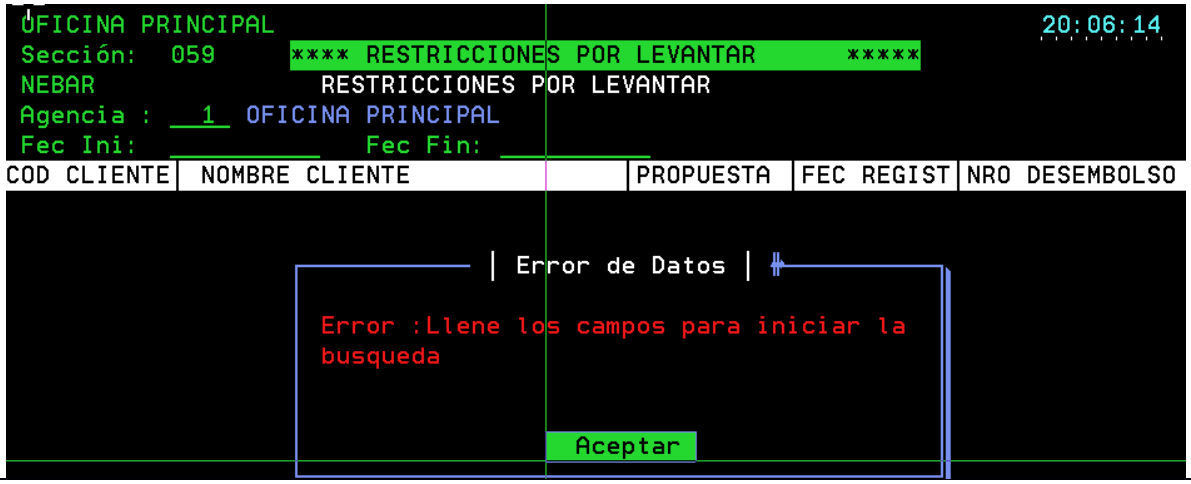
Presionamos la opción F8 y nos mostrará la siguiente pantalla.



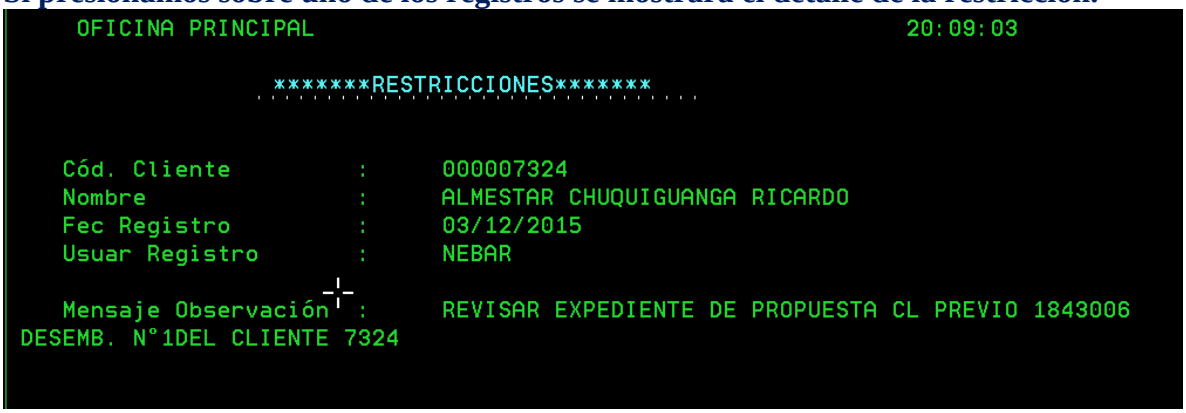
Llenamos los campos para realizar la búsqueda y presionamos la tecla "ENTER" para procesar la búsqueda, también la tecla F4 para solicitar la agencia a consultar.



También valida si los datos son correctos.

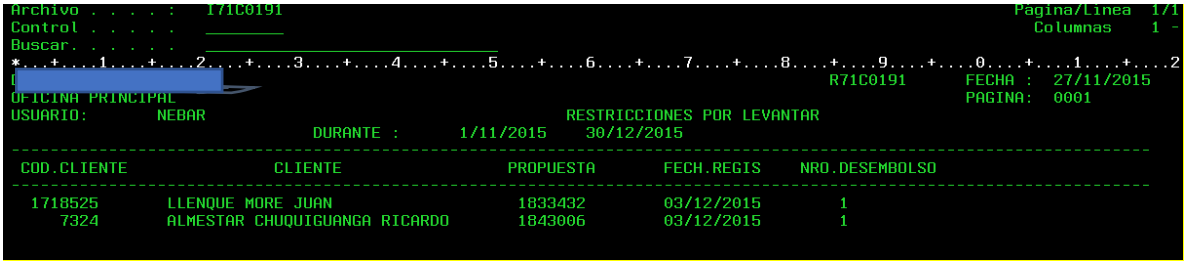


Si presionamos sobre uno de los registros se mostrará el detalle de la restricción.

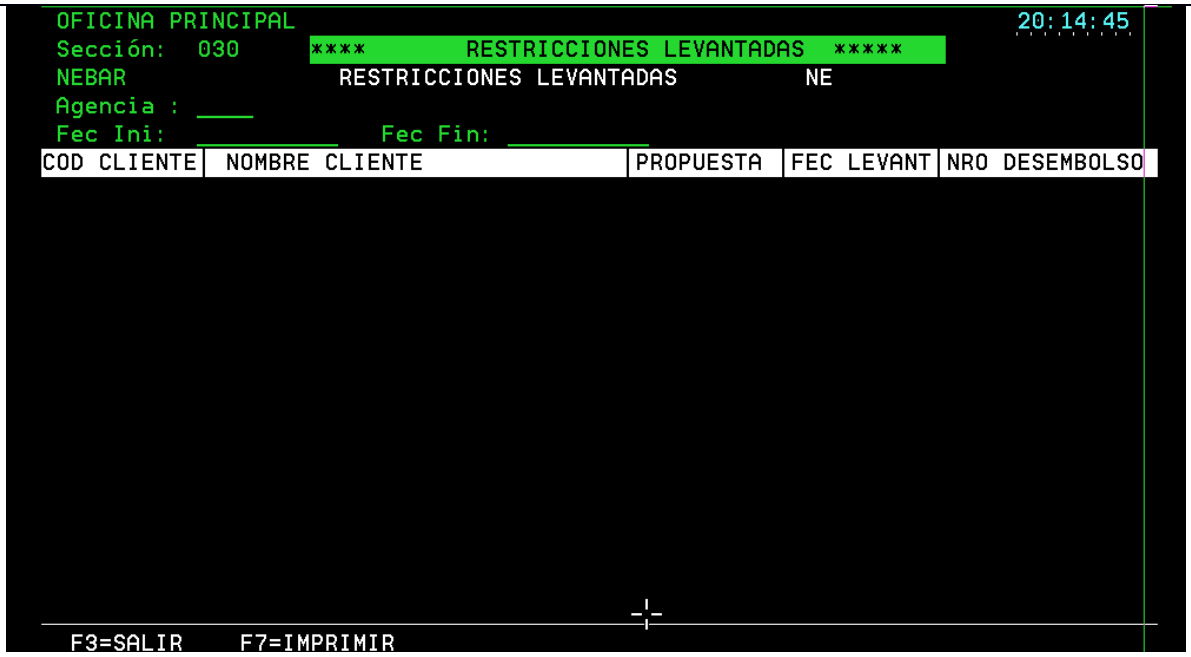


Si presionamos la tecla F7 podremos procesar el reporte.

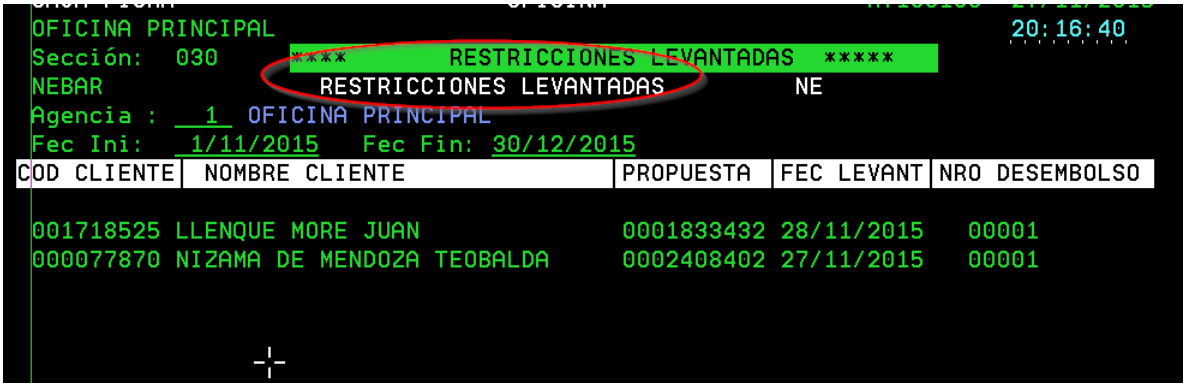
Ahora visualizamos su contenido.



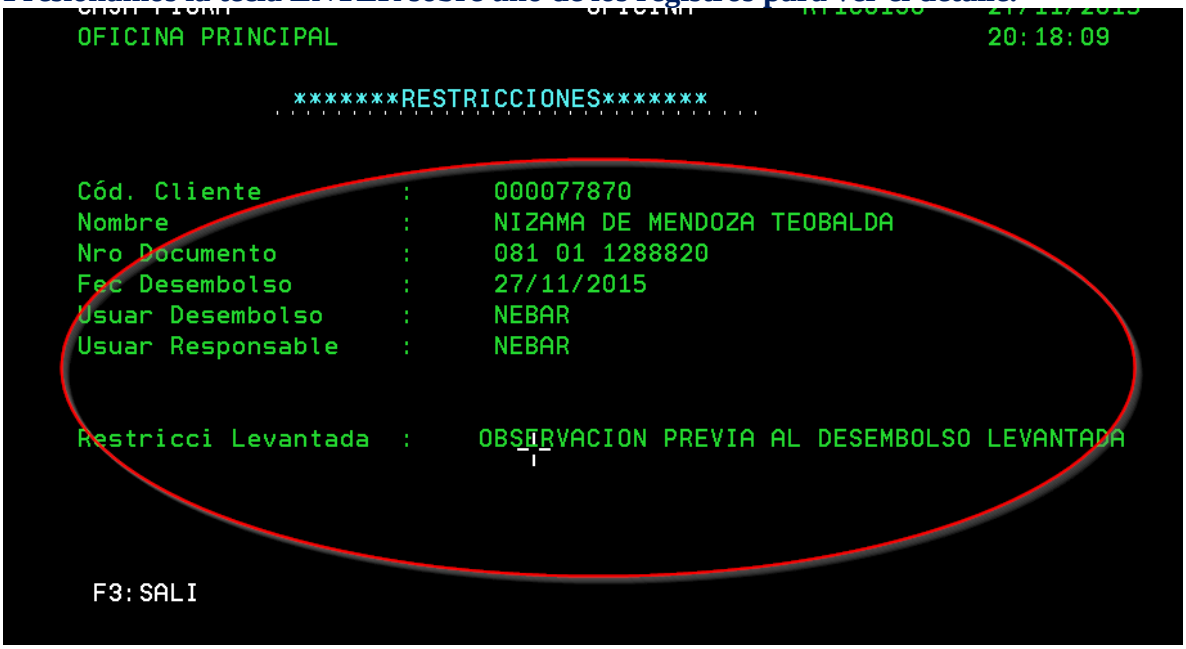
37) Reporte de restricciones levantadas entre fechas.



La ejecución es la misma que en el punto 7, sólo que en este el detalle de cada registro será distinto



Presionamos la tecla ENTER sobre uno de los registros para ver el detalle.



Presionamos le tecla F7 para procesar el reporte.

Visualizamos el contenido.

Sello y Firma del Desarrollador	**Sello y Firma del Desarrollador**

Anexo D: Código de Fuente del módulo M07U0002

```
 //-------------------------------------------------------------------*
 //            SOFTWARE FINANCIERO AUTOMATIZADO (SOFIA)               *
 //            ---------------------------------------                *
 // MODULO       : CREDITOS                                           *
 // APLICACION   : CHECK-LIST Desembolso de Préstamo                  *
 //-------------------------------------------------------------------*
 // AUTOR        : Nelson Abel Barranzuela Iman                       *
 // FECHA CREAC. : 19/08/2015                                         *
 //-------------------------------------------------------------------*
 // MODIFICADO POR :                         FECHA MODIF.:            *
 // =================                        =============            *
 //
 //-------------------------------------------------------------------*
 // COMENTARIO   :                                                    *
 // ==========                                                        *
 //-------------------------------------------------------------------*
 // Opciones de control                                               *
 //-------------------------------------------------------------------*
h AlwNull(*usrctl)
hdatfmt(*EUR) timfmt(*EUR)
hNOMAIN
hBNDDIR('SRVPBND')

d/DEFINE PROPUESTA_H_DEFINED
d/COPY QRPGCPYLE,R07U0002

d/DEFINE UTILITYSQLRPG_H_DEFINED
d/COPY QRPGCPYLE,R01U0001
 *********************************************************************
 * PROTOTYPES                                                        *
 *********************************************************************


 ********************************************************************
 * getDataChecklist: Retorna preguntas y respuestas de Check-List*
 * propuestaNumber = (Input) número de propuesta                    *
 * tipoChkList = (Input) tipo de checklist                          *
 *             = (Output) Vector de Datos                           *
 * tipoPersona(Input) = puede ser :                                 *
 *                  'N' :  NATURA                                   *
 *                  'J' : JURÍDICA                                  *
 ********************************************************************
P getDataChecklist...
P                 B                   Export
d                 PI                  likeDs(ChekList_Data_t) dim(99)
d  propuestaNumber...
d                               10P 0 Const
d  tipoChkList                   1P 0 Const
d  tipoPersona                   1A   Const
d
d  ArrdataCheckList...
d                 ds                  likeDs(ChekList_Data_t) dim(99)
d
d  dataCheckListAux...
d                 ds                  likeDs(ChekList_Data_t)
```

```
d
d  count          s              2  0
d
  /Free
    //DIRECTIVAS SQL
    Exec SQL Set Option Commit=*None, Naming = *Sys;
    Exec SQL Set Path=*Libl;

    count = 0;

    Exec SQL
      Declare ChkList_Cursor CURSOR FOR
      SELECT T.*, C.DESLAP, C.VALENT FROM
      (SELECT A.DOCCHKLIST_TIPOCATEGORIA AS KIND_CATEGORY,
      Row_Number() over() AS INDICE, B.DOCCHKLIST_CODIGODOCCHKLIST
      ,A.DOCCHKLIST_DESCRIPCION, (CASE WHEN C.DETEXPPROP_RPTA
      IS NULL THEN 0 WHEN C.DETEXPPROP_RPTA = 4 THEN 0
      ELSE 1 END) AS ANSWER_SYSTEM, A.DOCCHKLIST_ACEPTARESPUESTANO
      FROM F8051 A INNER JOIN F8052 B
      ON A.DOCCHKLIST_CODIGODOCCHKLIST = B.DOCCHKLIST_CODIGODOCCHKLIST
      LEFT JOIN F8053 C ON
      B.DOCCHKLIST_CODIGODOCCHKLIST = C.DOCCHKLIST_CODIGODOCCHKLIST
      AND C.PROPNUMERO = :propuestaNumber
      WHERE B.CHKLIST_TIPOCHKLIST = :tipoChkList
      AND LOCATE(:tipoPersona,TRIM(A.DOCCHKLIST_PERSONA)) > 0
      ) T, F6209 C
      WHERE C.DIGIDE = 643 AND CODTAB = VARCHAR(T.KIND_CATEGORY)
      for fetch only;

   Exec Sql Open ChkList_Cursor;
   Exec Sql Fetch ChkList_Cursor Into: dataCheckListAux;

   Dow SqlCode = 0;
     count += 1;

   ArrdataCheckList(count).kindCategory = dataCheckListAux.kindCategory;
   ArrdataCheckList(count).indice = dataCheckListAux.indice;
   ArrdataCheckList(count).codigoDoc = dataCheckListAux.codigoDoc;
   ArrdataCheckList(count).descripcion = dataCheckListAux.descripcion;
   ArrdataCheckList(count).systemanswer = dataCheckListAux.systemanswer;
   ArrdataCheckList(count).answerAllowNo =
                                        dataCheckListAux.answerAllowNo;
   ArrdataCheckList(count).deslap = dataCheckListAux.deslap;
   ArrdataCheckList(count).valorEntero = dataCheckListAux.valorEntero;

   Exec Sql Fetch ChkList_Cursor Into: dataCheckListAux;
   check_SQLState();
   EndDo;
   Exec SQL Close ChkList_Cursor;
   Return ArrdataCheckList;

  /End-Free
P                 E

 ********************************************************************
 *  @Procedure   : saveTipoExpedCheckList
 *  @Description : Procedimiento que graba TIPO DE EXPEDIENTE CHECK LIST
 *  @Input       : rowTipoExpChkList Tipo de Check-List a registrar
 *  @Return      : True si Operación fue existosa, False en caso
```

```
 *                  contrario
 ****************************************************************************
P saveTipoExpedCheckList...
P                 B                   Export
D                 PI             N
D  rowTipoCheckListAux...
D                                     Likeds(rowTipoExpChkList) Const
D
D Ok              s               N   Inz(*Off)
D
  /FREE

    //---Guardar tipo de CheckList
    EXEC SQL
      INSERT INTO F8053 VALUES (:rowTipoCheckListAux);
      check_SQLState();
    Ok = *On;
    return Ok;

  /END-FREE
P                 E

 ****************************************************************
 * @Procedure   : deleteTipoExpedCheckList
 * @Description : Elimina un tipo de Check-List
 * @Input       : rowTipoExpChkList Tipo de Check-List a registrar
 * @Return      : True si Operación fue existosa,
 *                False en caso contrario
 ****************************************************************
p deleteTipoExpedCheckList...
p                 B                   Export
D                 PI             n
d  rowTipoCheckListAux...
d                                     LikeDs(rowTipoExpChkList) Const
d
d  Ok             s               n   Inz(*Off)
d
  /FREE
      // Borrar tipo de Check-List
      Exec Sql
      DELETE FROM F8053 WHERE COEXPRO = :rowTipoCheckListAux.COEXPRO
      AND PROPNRO = :rowTipoCheckListAux.PROPNRO
      AND TIPEXP = :rowTipoCheckListAux.TIPEXP;
      check_SQLState();
      OK = *On;

      Return Ok;
  /END-FREE
P                 E

 ************************************************************
 * saveExpedCheckListPropuesta() : Registra un Expediente de  *
 *                                 Check-List.                *
 * rowExpedChkListPropuesta(input) : estructura con los datos *
 *                                   a registrar              *
 * isOk(Output) : Flag, True operación correcta, False        *
 *                 caso contrario                             *
 ************************************************************
P saveExpedCheckListPropuesta...
```

```
P                 B                   Export
d                 PI             N
d rowExpedChkListProP...
d                                     likeds(rowExpedChkListPropuesta)
d isOk            s               N   Inz(*Off)
d
  /Free
     Exec Sql
     Insert Into F8054 Values(:rowExpedChkListProP);
     check_SQLState();
     isOk = *on;
     Return isOk;

  /End-Free
P                 E

 ******************************************************************
 * updateExpedCheckListPropuesta()                                *
 * rowExpedChkListProP(Input) : Estructura de datos de expediente *
 *                              de Check-List                     *
 * successfully(Output) : Flag, True todo correcto, False         *
 *                        caso contrario                          *
 ******************************************************************
P updateExpedCheckListPropuesta...
P                 B                   Export
d                 PI             n
d rowExpedChkListProP...
d                                     likeDs(rowExpedChkListPropuesta)
 * Stand Alone Variables
d successfully    s                n   Inz(*Off)
d queryStringUpd  s              500a
d fieldsToChange  s              400a   Varying
  /FREE

     queryStringUpd = 'UPDATE F8054 SET ';
     If (rowExpedChkListProP.NUMDES <> *Zeros);
         fieldsToChange = ' NUMDES = '
           + %char(rowExpedChkListProP.NUMDES);
     EndIf;
     If (rowExpedChkListProP.TIPCHK <> *Zeros);
         If (%len(fieldsToChange) > *Zeros);
           fieldsToChange = %trim(fieldsToChange) +  ', TIPCHK = '
           + %char(rowExpedChkListProP.TIPCHK);
         Else;
           fieldsToChange = %trim(fieldsToChange) + ' TIPCHK = '
           + %char(rowExpedChkListProP.TIPCHK);
         EndIf;
     EndIf;
     If (rowExpedChkListProP.FLGSIT <> *Zeros) ;
         If (%len(fieldsToChange) > 0);
           fieldsToChange = %trim(fieldsToChange) + ', FLGSIT = '
           + %char(rowExpedChkListProP.FLGSIT);
         Else ;
             fieldsToChange = %trim(fieldsToChange) + ' FLGSIT = '
           + %char(rowExpedChkListProP.FLGSIT);
         EndIf;
     EndIf;
     If (rowExpedChkListProP.FLGEST <> 0) ;
        If (%len(fieldsToChange) > 0);
```

```
      fieldsToChange = %trim(fieldsToChange) + ', FLGEST = '
      + %char(rowExpedChkListProP.FLGEST);
   Else ;
        fieldsToChange = %trim(fieldsToChange) + ' FLGEST = '
      + %char(rowExpedChkListProP.FLGEST);
   EndIf;
EndIf;
If (rowExpedChkListProP.COMREV <> *Blanks) ;
   If (%len(fieldsToChange) > 0);
      fieldsToChange = %trim(fieldsToChange) +', COMREV = '
      + QUOTE + rowExpedChkListProP.COMREV + QUOTE;
      Else ;
        fieldsToChange =%trim(fieldsToChange) + ' COMREV = '
      + QUOTE + rowExpedChkListProP.COMREV + QUOTE;
    EndIf;
EndIf;
If (rowExpedChkListProP.USRREV <> *Blanks);
   If (%len(fieldsToChange) > 0);
     fieldsToChange = %trim(fieldsToChange) + ', USRREV = '
     + QUOTE + rowExpedChkListProP.USRREV + QUOTE;
   Else ;
     fieldsToChange = %trim(fieldsToChange) +  ' USRREV = '
     + QUOTE + rowExpedChkListProP.USRREV + QUOTE;
   EndIf;
EndIf;
If (%char(rowExpedChkListProP.FECREV) <> DEFAULT_TIMESTAMP);
   If (%len(fieldsToChange) > 0);
      fieldsToChange =%trim(fieldsToChange) + ', FECREV = '
      + QUOTE + %char(rowExpedChkListProP.FECREV) + QUOTE;
   Else ;
      fieldsToChange = %trim(fieldsToChange)+ ' FECREV = '
      + QUOTE + %char(rowExpedChkListProP.FECREV) + QUOTE;
   EndIf;
EndIf;
If (rowExpedChkListProP.USRLEV <> *Blanks);
   If (%len(fieldsToChange) > 0);
      fieldsToChange = %trim(fieldsToChange) + ', USRLEV = '
      + QUOTE + rowExpedChkListProP.USRLEV + QUOTE;
   Else ;
      fieldsToChange =%trim(fieldsToChange) + ' USRLEV = '
      + QUOTE + rowExpedChkListProP.USRLEV + QUOTE;
    Endif;
EndIf;
If (%char(rowExpedChkListProP.FECLEV) <> DEFAULT_TIMESTAMP);
   If (%len(fieldsToChange) > 0);
      fieldsToChange = %trim(fieldsToChange) + ', FECLEV = '
      + QUOTE +  %char(rowExpedChkListProP.FECLEV) + QUOTE;
   Else ;
        fieldsToChange = %trim(fieldsToChange) + ' FECLEV = '
      + QUOTE + %char(rowExpedChkListProP.FECLEV) + QUOTE;
    EndIf;
EndIf;
If (rowExpedChkListProP.USRMOD <> *Blanks);
   If (%len(fieldsToChange) > 0);
     fieldsToChange = %trim(fieldsToChange) + ', USRMOD = '
     + QUOTE + rowExpedChkListProP.USRMOD + QUOTE;
   Else ;
     fieldsToChange = %trim(fieldsToChange) + ' USRMOD = '
     + rowExpedChkListProP.USRMOD;
```

```
        EndIf;
     EndIf;
     If (%char(rowExpedChkListProP.FECMOD) <> DEFAULT_TIMESTAMP);
        If (%len(fieldsToChange) > 0);
           fieldsToChange = %trim(fieldsToChange) + ', FECMOD = '
           + QUOTE + %char(rowExpedChkListProP.FECMOD) + QUOTE;
        Else ;
             fieldsToChange = %trim(fieldsToChange) + ' FECMOD = '
           + QUOTE + %char(rowExpedChkListProP.FECMOD) + QUOTE;
        EndIf;
     EndIf;
     queryStringUpd = %trim(queryStringUpd) + ' '
              + %trim(fieldsToChange) + ' WHERE '
              + 'PROPNRO = ' + %char(rowExpedChkListProP.PROPNRO)
              + ' AND TIPCHK = ' + %char(rowExpedChkListProP.TIPCHK);
    //Para ejecutar sentencia, name statement should be unique
    Exec SQL
        EXECUTE immediate :queryStringUpd;
     successfully = *On;
     Return successfully;

  /END-FREE
P                 E
 ****************************************************************
 * existExpedCheckListProp() : Verificar la existencia de un  *
 *                             expediente de Check-List       *
 * numeroPropuesta(input) : numero de propuesta               *
 * tipoCheckList(input) : tipo de checklist                   *
 * estadoCheckList(input) : estado del checklist 0 ó 1        *
 * return(output) :  True si existe, False caso contrario     *
 ****************************************************************
p existExpedCheckListProp...
p                 B                   Export
d                 PI              N
d  numeroPropuesta...
d                                19P 0 Const
d  tipoCheckList                  1P 0 Const
d  estadoCheckList...
d                                 1P 0 Const
d  existExp       s                N   Inz(*off)
d  countExpedts   s               14  0 Inz(0)
d
 /FREE
   Exec Sql
     Set :countExpedts = (select count(*) as total from F8054
                          where PROPNRO = :numeroPropuesta
                          and TIPCHK = :tipoCheckList
                          and  FLGEST = :estadoCheckList);
    check_SQLState(); //verificamos si todo va bien
    If ( countExpedts > *zero ) ;
       existExp = *on;
    EndIf;
    Return existExp;

 /End-Free
p                 E
 ****************************************************************
 * getExpedCheckListDto() : retorna una estructa de datos con la*
 *                          información necesaria para un        *
```

```
 *                          expediente de Check-List.            *
 * nroPropuesta(input) : número de propuesta.                    *
 * tipoCheckListPropuesta(input) : Tipo de Check-List.           *
 ****************************************************************
p getExpedCheckListDto...
p                 B                   Export
d                 PI                  likeDs(expedChkListPropuestaDto)
d nroPropuesta                  19P 0 Const
d tipoCheckListPropuesta...
d                                1P 0 Const
d dsExpedCkListDto...
d                 ds                  likeDs(expedChkListPropuestaDto)
  /Free

     Exec Sql
       Select COEXPRO,FLGSIT,COALESCE(COMREV,'') Into :dsExpedCkListDto
       From F8054 Where PROPNRO = :nroPropuesta
       And TIPCHK = :tipoCheckListPropuesta;
       check_SQLState();

     Return dsExpedCkListDto;
  /End-Free
p                 E

 //****************************************************
 //getNextValueFromCorrelativos(): Obtiene siguiente  *
 //            correlativo para código Check-List     *
 //codigo(input) : último código de Check-List        *
 //flagUpdateValue(input) : Bandera que que indica si *
 //            se actualiza dicho para parámetro      *
 //****************************************************
p getNextValueFromCorrelativos...
p                 B                   Export
d                 PI            19P 0
d  codigo                       19P 0 Const
d  flagUpdateValue...
d                                 N   Const
d
d  nextValue      s             19P 0
d
  /Free

    Exec Sql
      Set :nextValue = ( SELECT (TABCORRELATIVO +1 ) AS VALOR
                         FROM CORRELATIVOS
                         WHERE TABCODIGO = :codigo );

      If (flagUpdateValue);
         updateCorrelativos(codigo:nextValue);
      EndIf;

      return nextValue;
  /End-Free
p                 E

 ****************************************************************
 * updateCorrelativo()
 * tabCodig(input) :
 * newTabCorrelativo(input) :
```

```
 * return :
 ******************************************************************
p updateCorrelativos...
p                 B                   Export
d                 PI             N
d tabCodigoY                     19P 0 Const
d newTabCorrelativo...
d                                19P 0 Const
d isOk            s               N   Inz(*off)
d
   /FREE
     Exec SQL
         UPDATE CORRELATIVOS
         SET TABCORRELATIVO = :newTabCorrelativo
         WHERE TABCODIGO = :tabCodigoY;
         check_SQLState();
     isOk = *ON;
     return isOK;
   /END-FREE
p                 E

 ******************************************************************
 * getDesLarFromDigide(): Retorna descripción larga de un digide
 * table(input) : tabla de digides
 * digide(input) : número de digide a consultar
 * deslar(output) : Descripción larga
 ******************************************************************
p getDesLarFromDigide...
p                 B                   Export
d                 PI            80A   varying
d table                         10A
d digide                         3P 0
d codtab                         6A
d
d queryString     s             500A
d descripcionLarga...
d                 s             80A   Varying inz(*blanks)
d
  /FREE
     queryString = 'SELECT DESLAP FROM '
         + %trim(table) + ' WHERE DIGIDE = '
         + %char(digide) + ' AND CODTAB = '+ QUOTE + codtab + QUOTE;

     Exec SQL PREPARE MySQLStatement FROM :queryString;

     Exec SQL Declare descripLarCS Cursor WITH HOLD for
                 MySQLStatement;

     Exec SQL Open descripLarCS;

     dou SQLSTATE = '02000';
         Exec SQL
            Fetch next from descripLarCS into :descripcionLarga;
         If SQLState = '02000' or SQLCode < *zeros;
          Leave;
         endif;
     Enddo;

     return descripcionLarga;
```

```
  /END-FREE
p                 E

 ******************************************************************
 * getNumberMessagePropuesta() : Obtiene el número de mensajes de *
 *                      Check-Lis que posee una propuesta         *
 *                      de desembolso de crédito.                 *
 * nroPropuesta = (input) numero de propuesta                     *
 * nroDesembolso = (input) tipo de checklist                      *
 *  nroMessagePropuesta  = (output) Cantidad de mensajes          *
 *                        de Check-List de una propuesta          *
 ******************************************************************
P getNumberMessagePropuesta...
P                 B                   Export
 * Interfaz
D                 PI             5P 0
D  nroPropuesta                 10P 0 Const
D  nroDesembolso                 5P 0 Const
 *
D  nroMessagePropuesta...
D                 s              5P 0 INZ(0)
  /FREE

      Exec SQL
      SELECT COUNT(*) INTO : nroMessagePropuesta FROM F7226
      WHERE REPCRE= :nroPropuesta AND NUMDES= :nroDesembolso;
      check_SQLState();

      return nroMessagePropuesta;

  /END-FREE
P                 E

 ******************************************************************
 * saveMessageDesembolso() : Guarda un registro en Archivo F7226  *
 * dsMessageDesembolso(input) : estructura de datos a insertar    *
 * flagSaveOk(Output) : Flag, True si tod fue correcto, False     *
 *                      caso contrario                            *
 ******************************************************************
p saveMessageDesembolso...
p                 B                   Export
d                 PI              n
d dsMessageDesembolso...
d                                     likeds(rowMessageDesembolso) const
d flagSaveOk      s               n   Inz(*off)
d
  /FREE

    Exec SQL
      INSERT INTO F7226 VALUES(:dsMessageDesembolso);
      flagSaveOk = *On;
      check_SQLState();
      Return flagSaveOk;

  /END-FREE
p                 E

 ******************************************************************
```

```
 * updateMessageDesembolso()  : Actualiza archivo F7226
 * messageMessageToUpd = (Input) Mensaje de CheckList a Actualizar
 * updOk = (Output) Flag, True si todo fue Correcto, False
 *         caso contrario
 ******************************************************************
p updateMessageDesembolso...
p                 B                   Export
d                 PI              N
d messageMessageToUpd...
d                                     likeds(rowMessageDesembolso) const
d updOk           s               N   Inz(*off)
d
  /FREE
      Exec Sql
        update F7226 set USULEV = :messageMessageToUpd.USULEV,
          DIALEV = :messageMessageToUpd.DIALEV,
          MESLEV = :messageMessageToUpd.MESLEV,
          AÑOLEV = :messageMessageToUpd.AÑOLEV,
          OBSERV = :messageMessageToUpd.OBSERV
          where REPCRE = :messageMessageToUpd.REPCRE
          AND NUMDES = :messageMessageToUpd.NUMDES AND
          MENSAJ LIKE 'CHECK LIST DURANTE DESEMB.%';
          check_SQLState(); //Make sure that all is well
      updOk = *on;
      return updOk;

  /END-FREE
p                 E
 ******************************************************************
 * getClienteDto() : retorna los datos más importantes de un cliente
 *                   CMAC PIURA en una estructura de datos.
 * codigoCli(input) : Código del cliente
 * clienteDto(output) : Estructura de datos con los datos del cliente
 ******************************************************************
p getClienteDto   B                   Export
d getClienteDto   PI                  likeds(cliente_t)
d  codigoCli                     9P 0 Const
d
d  clienteDto     ds                  likeDs(cliente_t)
d
  /FREE
      Exec SQL
      SELECT TRIM(NOMBR1) , TRIM(NOMBR2), TRIM(APEPAT),
      TRIM(APEMAT), TRIM(CLAPER) AS TIPO_PERSONA,
      CODCLI, AGEAPE, TRIM(NOMBCL),
      (CASE WHEN TIPDOC = '' THEN '7' ELSE TIPDOC END) AS TIP_DOI,
      (CASE WHEN CLAPER <> 'N' THEN TRIM(SIGLAS)
      ELSE DOCUME END) AS DOI into: clienteDto FROM F5101
      WHERE CODCLI = :codigoCli;
      check_SQLState();

      return clienteDto;

  /END-FREE
p                 E

 ******************************************************************
 * getPropuestaEstado() : retorna el estado de la propuesta
 * number: numero de propuesta
```

```
 * numeroDesembolso : número de desembolso
 * estadoDesembolso : estado del desembolso
 ****************************************************************
p getPropuestaEstado...
p                 B                   Export
d getPropuestaEstado...
d                 PI             1P 0
d  number                       10P 0 Const
d  numeroDesembolso...
d                                5P 0 Const
d  estadoDesembolso...
d                 s              1P 0
d*
  /FREE
      Exec SQL
      Set :estadoDesembolso = (SELECT ESTREP FROM F7220
      WHERE REPCRE = :number AND NUMDES = : numeroDesembolso);
      check_SQLState();

    return estadoDesembolso;

  /END-FREE
p                 E
 ////////////////////////////////////////////////////////
 // getDatosPagarePorPropuesta()
 // nroPropuesta(input): numero de propuesta
 // nroDesembolso(input) : numero de desembolso
 // dsDatosPagare(output) : datos de pagare
 ////////////////////////////////////////////////////////
p getDatosPagarePorPropuesta...
p                 B                   Export
d                 PI                  likeds(pagare_t)
d  nroPropuesta                 10P 0 Const
d  nroDesembolso                 5P 0 Const
d*
d dsDatosPagare   ds                  likeds(pagare_t) Inz
d
  /FREE
     Exec SQL
       SELECT A.SERVIC, A.MONEDA, A.NUMOPE, A.DIADES, A.MESDES, A.AÑODES
       into :dsDatosPagare
       FROM F7101 A INNER JOIN F7220 B ON A.REPCRE = B.REPCRE
       WHERE A.REPCRE = :nroPropuesta AND B.NUMDES = :nroDesembolso;
       check_SQLState();
     return dsDatosPagare;
  /END-FREE
p                 E
 ////////////////////////////////////////////////////////
 // getUsuarioDesembolso(): Retorna el usuario que realizo
 //                         el desembolso, para ello realiza
 //                         la búsqueda en los historicos
 // servici(input) : servicio
 // moneda(input) : moneda
 // cuenta(cuenta) : cuenta
 ////////////////////////////////////////////////////////
P getUsuarioDesembolso...
P                 B                   EXPORT
d                 PI             10A
d  transaccion                   4P 0 Const
```

```
d  servicio                      3P 0 Const
d  moneda                        2P 0 Const
d  cuenta                        7P 0 Const
d*
d usuarioDesembolso...
d                 s             10A   Inz('')
d nullInd         s              5I 0
  /FREE
    Exec SQL
      Set :usuarioDesembolso:nullInd = (SELECT COALESCE(USUARI, '')
        FROM F2405
       WHERE TRANSA = :transaccion AND SERVIC = :servicio
       AND MONEDA = :moneda AND CUENTA = :cuenta AND NROENC = 0
       AND REVERS = 0 ORDER BY HORA DESC FETCH FIRST 1 ROWS ONLY);
      check_SQLState();

      if usuarioDesembolso = '';
         Exec SQL
         Set :usuarioDesembolso:nullInd = (SELECT COALESCE(USUARI, '')
         FROM F6103
         WHERE TRANSA = :transaccion AND SERVIC = :servicio AND
         MONEDA = :moneda AND CUENTA = :cuenta AND NROENC = 0
         ORDER BY HORA DESC FETCH FIRST 1 ROWS ONLY);
         check_SQLState();
      endif;

      if usuarioDesembolso = '';
         Exec SQL
         Set :usuarioDesembolso:nullInd = (SELECT COALESCE(USUARI, '')
         FROM F6104
         WHERE TRANSA = :transaccion AND SERVIC = :servicio
         AND MONEDA = :moneda AND CUENTA = :cuenta
         AND NROENC = 0  ORDER BY HORA DESC FETCH FIRST 1 ROWS ONLY);
        check_SQLState();
      endif;

      if usuarioDesembolso = '';
         Exec SQL
         Set :usuarioDesembolso:nullInd = (SELECT COALESCE(USUARI, '')
         FROM A6104
         WHERE TRANSA = :transaccion AND SERVIC = :servicio
         AND MONEDA = :moneda AND CUENTA = :cuenta
         AND NROENC = 0  ORDER BY HORA DESC FETCH FIRST 1 ROWS ONLY);
        check_SQLState();
      endif;

      if usuarioDesembolso = '';
         Exec SQL
         Set :usuarioDesembolso:nullInd = (SELECT COALESCE(USUARI, '')
         FROM A6103
         WHERE TRANSA = :transaccion
         AND SERVIC =:servicio AND MONEDA = :moneda AND CUENTA = :cuenta
         AND NROENC = 0  ORDER BY HORA DESC FETCH FIRST 1 ROWS ONLY);
        check_SQLState();
      endif;

      return usuarioDesembolso;
  /END-FREE
P                 E
```

```
 ////////////////////////////////////////////////////
 // getDetailMessageRestriccion() :
 // propuestaNumero(input) : número de propuesta
 // desembolsoNumero(input) :número de desembolso
 // dsDetailMessResric(output) : datos del mensaje
 ////////////////////////////////////////////////////
P getDetailMessageRestriccion...
P                 B                   Export
d                 PI                  likeds(detailMessageRestric_t)
d  propuestaNumero...
d                                10P 0 Const
d  desembolsoNumero...
d                                 5P 0 Const
d*
d  dsDetailMessResric...
d                 ds                  likeds(detailMessageRestric_t)
  /FREE
     Exec SQL
       SELECT USULEV,USUREG, MENSAJ, OBSERV into :dsDetailMessResric
       FROM F7226
       WHERE REPCRE = :propuestaNumero AND NUMDES = :desembolsoNumero
       AND  MENSAJ LIKE  'REVISAR EXPEDIENTE DE PROPUESTA CL PREVIO%';
       check_SQLState();

     return dsDetailMessResric;
  /END-FREE
P                 E
 ////////////////////////////////////////////////////////////
 // getEmployeByUserId() :
 // employeeIdUser(input) : Id de usuario
 // employeeDto(output) : datos de empleado
 ////////////////////////////////////////////////////////////
p getEmployeByUserId...
p                 B                   Export
d                 PI                  likeds(employee_t)
d  employeeIdUser...
d                                10A   Const
d*
d employeeDto     ds                  likeds(employee_t)
d
  /Free
     Exec SQL
        SELECT EMPCOD, NOMEAB, CODAGE, CODPUE into :employeeDto FROM
        F6701
       WHERE USUARI = :employeeIdUser;
       check_SQLState();

     return employeeDto;
  /End-Free
p                 E

 ////////////////////////////////////////////////////////////
 // getAgenciaByCodigo ():  Retorna datos de la agencia
 // agenciaCodigo(input) : codigo de agencia
 // agenciaDto(output) : datos de agencia
 ////////////////////////////////////////////////////////////
P getAgenciaByCodigo...
P                 B                   Export
d                 PI                  likeds(agencia_t)
d  agenciaCodigo...
```

```
d                                3P 0 Const
d*
d agenciaDto      ds                  likeds(agencia_t)
d
  /Free
      Exec Sql
        SELECT NOMAGE, NOMADM, CIUDAG into :agenciaDto FROM F2202
        WHERE AGENCI = :agenciaCodigo;
        check_SQLState();

     return agenciaDto;
  /End-Free
P                 E
 ////////////////////////////////////////////////////////
 // isUserAdminOfAgencia() :
 // idAdministrador(input) : id de usuario a evaluar
 // isAdmin(output) : true si es verdad, falso
 //                   en caso contrario
 ////////////////////////////////////////////////////////
P isUserAdminOfAgencia...
P                 B                   Export
d                 PI             N
d  idAdministrador...
d                               10A   Const
d*
d  counterValues  s               5P 0 inz(0)
d  isAdmin        s                N   inz(*off)
d
  /Free
     Exec SQL
     SELECT COUNT(*) into :counterValues FROM F6701
     WHERE USUARI = :idAdministrador AND  CODPUE IN (1, 2, 71);
     check_SQLState();
     if counterValues > 0;
        isAdmin = *on;
     endif;

     return isAdmin;
  /End-Free
P                 E
```

Anexo E: código de fuente del módulo M01U0001

```
//-----------------------------------------------------------------*
//            SOFTWARE FINANCIERO AUTOMATIZADO (SOFIA)             *
//            ---------------------------------------              *
// MODULO       : CREDITOS                                         *
// APLICACION   : UTILIDADES DE SISTEMA PARA MANEJO DE ERRORES     *
//                Y ALGUNOS COMANDOS DEL SISTEMA                   *
//-----------------------------------------------------------------*
// AUTOR        : Nelson Abel Barranzuela Iman(NEBAR)              *
// FECHA CREAC. : 22/10/2015                                       *
//-----------------------------------------------------------------*
// MODIFICADO POR :                        FECHA MODIF.:           *
```

```
 // =================                          =============        *
 //                                                                  *
 //------------------------------------------------------------------*
 // COMENTARIO   :                                                   *
 // ==========                                                       *
 //------------------------------------------------------------------*
 // Opciones de control                                              *
 //------------------------------------------------------------------*
H noMain option(*srcStmt: *noDebugIO)

d/DEFINE UTILITYSQLRPG_H_DEFINED
d/COPY QRPGCPYLE,R01U0001

 //**/ @desc pushLib . <br />
 // Agrega una libraria al inicio de la lista de
 // librerías
 // @Lib(input) : libreria a agregar
p pushLib         B                   export
d                 PI
d Lib                            10a   Const
d
  /FREE
     addLibLE(Lib:'*FIRST');
  /END-FREE
p                 E
 // **/desc popLib .<br/>
 // Remueve una libreria de la lista de librerias
 // Si ninguna libreria es pasada a esta función
 // o el valor especia '*FIRST' la primera libreria
 // es removida.
 // @Lib(input) : libreria a remover de la lista de librerias
 //   *FIRST - Remueve la primera en la lista de librerias
 //   library - Remueve "library" de la lista de librerias
 ////////////////////////////////////////////////////////////
P popLib          B                   Export
d popLib          PI
d  Lib                           10    Const Options(*nopass)
 *
d libPtr          s                 *
 *
d libData         ds                  based(libPtr)
d  nroLibs                       10I 0
d  libArr                        10    dim(25)
 *
d tempLib         s                10
d cmdToExecute    s               500A
d
  /Free
    if (%parms< 1) or (Lib = '*FIRST');
          libPtr = rtvLibl('*USER');
          if (libPtr <> *null) and (nroLibs > 0);
            tempLib = libArr(1);
          endif;
    else;
          tempLib = Lib;
    endif;

    cmdToExecute = 'RMVLIBLE LIB('
                   + %trim(tempLib) + ')';
```

```
    monitor;
      callp(e) Command(%trim(cmdToExecute):200);
       ON-ERROR;
    endmon;
    return;

  /End-Free
P                 E

 //**/ @desc verLib code. <br/>
 // Verifica que una libreria se encuentra en la lista
 // de librerias. El valor actual retornado sera el valor
 // de la posición actual de la librería en la lista del
 // tipo de librería que se pase como parametro.
 // Un valor menor que 1 indica que la libreria no se
 // encontró en la lista de librerías especificada.
 //
 // @library(input) : Library to verify.
 // @librariType(input) : El tipo de lista de libreria en la cual
 //                       buscar el valor referenciado en el parametro
 //                       @library.
 // *SYSTEM : Retorna la lista de las librerias del sistema
 // *PRODUCT: Retorna la lista de las librerias del producto
 // *CURRENT: Retorna la lista de libreria actual, por ejem. 'NEBAR'
 // *USER   : Retorna la lista de librerias del usuario.
 ////////////////////////////////////////////////
P existLibrary    B                   Export
d existLibrary    PI            10I 0
d  library                      10    Const
d  libraryTye                   10    Const
 *
d libraryData     ds                  Based(libraryPointer)
d   nroLibraries                10I 0
d   libraryArray                10     Dim(25)
 *
d position        s               2  0  Inz(0)
d
  /FREE

    libraryPointer = rtvLibl(libraryTye);

    if (libraryPointer = *null);
       return -1;
    endif;

    position = %lookup(%trim(library):libraryArray);

    return position;

  /END-FREE
P                 E

 //**/ @desc Check SQL status code. <br />
 //     Checks the status code of the previously executed SQL
 //     statement and, depending on the status, will send a message
 //     to the caller. <br />
 //     if the status is a warning (SQLSTATE='01nnn'/SQLCODE>0), a
 //     diagnostic message is sent to the caller <br />
 //     if the status is an error(SQLSTATE='03nnn'/SQLCODE<0), an
```

```
 //     escape message is sent to the caller <br />
 //     The function also indicates if "Row Not Found" is detected
 //     True if BAD error (but escape message has been sent, so
 //     caller should fail) <br >
 //     True if "Row not Found" - SQLSTATE='02nnn'/SQLCODE=100

 //    @category utility
 //    @category SQL
 //    @category error
P check_SQLState  B                   export
D                 PI              n
  // Standard API Error Data Structure
d APIError        DS                  qualified
d  bytesProvided                10i 0 inz(%size(APIError))
d  bytesAvail                   10i 0 inz(0)
d  msgId                         7a
d                                1a
d  msgData                     240a

   // Prototype for QMHSNDPM (Send Program Message) API
d sendProgramMessage...
d                 PR                  extPgm('QMHSNDPM')
d   messageID                    7a   const
d   messageFile                 20a   const
d   messageData                256a   const
d   messageDataLength...
d                               10i 0 const
d   messageType                 10a   const
d   callStackEntry...
d                               10a   const
d   callStackCount...
d                               10i 0 const
d   messageKey                   4a
d   errorCode                        likeds(APIError)

    // Work fields
d messageKey      s               4a
d messageType     s              10a
d messageText     s            1024a
d status          s                n

d                 DS
d  lastState                     5a
d  status_SQL                    2a   overLay(lastState)

 // Constants
d W_DIAGNOSTIC    C                   '*DIAG'
d W_EOF           C                   '02'
d W_ESCAPE        C                   '*ESCAPE'
d W_MSGF          C                   'QCPFMSG   *LIBL'
d W_MSGID         C                   'CPF9897'
d W_STACK_ENTRY   C                   '*'
d W_STACK_COUNT1  C                   1
d W_SUCCESS       C                   '00'
d W_WARNING       C                   '01'

  /FREE
    // Get last state
    exec SQL
```

```
       get diagnostics condition 1 :lastState = RETURNED_SQLSTATE;

       //All OK - just return
      if (status_SQL = W_SUCCESS);

        // EOF - return true - but no message
       elseIf (status_SQL = W_EOF);
         status = *on;

         //Warning - send Diagnostic message
       elseIf (status_SQL = W_WARNING);
         messageType = W_DIAGNOSTIC;
         exec SQL
           get diagnostics condition 1: messageText = MESSAGE_TEXT;

         //Anything else - Send a Escape message
       else;
         messageType = W_ESCAPE;
         exec SQL
          get diagnostics condition 1: messageText = MESSAGE_TEXT;
        status = *on;
      endif;

      if (messageType <> *blanks);
          dump(A);
          messageText = lastState + ' ' + messageText;
          sendProgramMessage( W_MSGID
                            : W_MSGF
                            : messageText
                            : %len(%trimr(messageText))
                            : messageType
                            : W_STACK_ENTRY
                            : W_STACK_COUNT1
                            : messageKey
                            : APIError);
      endif;
      return status;

  /END-FREE
p                 E

 //**/ @desc addLible. <br />
 // addLibLE() : Agrega una libreria a un programa en ejecucion
 // @lib(input optional) : libreria a agregar a lista de librerias.
 // @post(input optional) :posición a ubicar la libreria
 //  *FIRST -Ubica la libreria al inicio de la lista de librerias.
 //  *LAST  -Ubica la libreria al final de las lista de librerias.
 //  *AFTER -Ubica la libreria después de la libreria referenciada
 //             en el parametro RefLib.
 //  *REPLACE -Reemplaza la libreria referencia en el parametro
 //            RefLib.
 // @RefLib (Optional): Nombre de la libreria a referenciar cuando
 //                   *AFTER, *BEFORE, o *REPLACE se especifica
 //                  en el parametro Pos.
 ////////////////////////////////////////////////////////////////
p addLibLE        B                   Export
d                 PI
d  lib                           10    Const
d  pos                            8    Const Options(*NOPASS)
```

```
d  refLib                        10    Const Options(*NOPASS)
d
d cmdToExecute    s             500A
d
   /FREE
     cmdToExecute = 'ADDLIBLE LIB(' + %trim(lib) +')';

     if (%parms > 2) and ((pos = '*BEFORE') or (pos = '*AFTER')
       or (pos = '*REPLACE'));
       cmdToExecute = %trim(cmdToExecute) + ' ' + 'POSITION('
              +  %trim(pos) + ' ' + %trim(refLib) + ') ';
      else;
        if (%parms > 1);
           cmdToExecute = %trim(cmdToExecute) + ' ' + 'POSITION('
                   + %trim(pos) + ')';
        endif;
     endif;

     monitor;
       callp(e) Command(%trim(cmdToExecute):200);
       ON-ERROR;
     ENDMON;
     return;
   /END-FREE
p                 E

 //**/ @desc rtvLibl
 // rtvLibl : recupera los datos de la Lista
 // de bibliotecas y retorna la información como un puntero a la
 // estructura de datos que contiene la información de la librería.
 // Si el puntero que retorna contiene el valor de nulo, un error
 // ha ocurrido.
 ////////////////////////////////////////////////////////////////////
P rtvLibl         B                   Export
d rtvLibl         PI              *
d   libType                      10    Const
 * Defining prototype
d QUSRJOBI        PR                  ExtPgm('QUSRJOBI')
d   rcvVar                    32767A   Options(*varsize)
d   rcvVarLen                    10I 0 Const
d   Format                        8A   Const
d   QualJob                      26A   Const
d   InternalId                   16A   Const
d   ErrorCode                 32767A   options(*varsize: *nopass)
d   Reset                         1A   options(*nopass)
 *
d  dataToRtv      ds
d    rtvSysLibs                  10I 0 Overlay(dataToRtv:65)
d    rtvPrdLibs                  10I 0 Overlay(dataToRtv:69)
d    rtvCurLibs                  10I 0 Overlay(dataToRtv:73)
d    rtvUsrLibs                  10I 0 Overlay(dataToRtv:77)
d    rtvData                    319    Overlay(dataToRtv:81)
 *
d SysData         ds                  Static
d  nroSysLibs                    10I 0
d  SysArr                        10    Dim(25)
 *
d PrdData         ds                  Static
d  nroPrdLibs                    10I 0
```

```
d  PrdArr                        10    Dim(25)
 *
d CurData         ds                   Static
d  nroCurLibs                    10I 0
d  CurrArr                       10    Dim(25)
 *
d UsrData         ds                   Static
d  nroUsrLibs                    10I 0
d  UsrArr                        10    Dim(25)
 * Variables
d rtvLen          s              10I 0 Inz(400)
d rtvFmt          s               8    Inz('JOBI0700')
d rtvJobName      s              26    Inz('*')
d rtvID           s              16    Inz(*blanks)
 * Other variables
d x               s              10I 0
d y               s              10I 0
d
  /Free

      QUSRJOBI(dataToRtv:rtvLen:rtvFmt:rtvJobName:rtvID);

      y = 1;
      nroSysLibs = rtvSysLibs;
      nroPrdLibs = rtvPrdLibs;
      nroCurLibs = rtvCurLibs;
      nroUsrLibs = rtvUsrLibs;

     select;
     when (libType = '*SYSTEM');

      for x = 1 to nroSysLibs;
        SysArr(x) = %subst(rtvData:y:10);
        y = (y+11);
      endfor;

     return %addr(SysData);

     when (libType = '*PRODUCT');
          y = (y + (nroSysLibs * 11));

       for x = 1 to nroPrdLibs;
         PrdArr(x) = %subst(rtvData:y:10);
         y = y + 11;
       endfor;

     return %addr(PrdData);

     when (libType = '*CURRENT');
        y = (y + ((nroSysLibs + nroPrdLibs)*11));

       for x = 1 to nroCurLibs;
        CurrArr(x) = %subst(rtvData:y:10);
        y = y + 11;
       endfor;

     return %addr(CurData);

      when (libType = '*USER');
```

```
         y = (y + ((nroSysLibs + nroPrdLibs
               + nroCurLibs)*11));

        for x = 1 to nroUsrLibs;
           UsrArr(x) = %subst(rtvData:y:10);
           y = y + 11;
        endfor;

         return %addr(UsrData);

       other;
       return *null;

       endsl;

       return *null;

  /End-Free
P                 E
 //**/ isValidDate
 // isValidDate()
 //
 ////////////////////////////////////////////////
p isValidNumericDate...
p                 B                   Export
d                 PI              n
d  numericDate                    8  0 Const
d  formatDate                     4A   Const
d*
d
  /FREE

       select ;
         when (%trim(formatDate) = '*EUR');
            Test(DE) *EUR numericDate;
            if (not %error) ;
             return *on;
            endif;
         when (%trim(formatDate) = '*ISO');
            Test(DE) *ISO numericDate;
            if (not %error) ;
              return *on;
            endif;
         when (%trim(formatDate) = '*USA');
            Test(DE) *USA numericDate;
            if (not %error) ;
             return *on;
         endif;
         when (%trim(formatDate) = '*JIS');
            Test(DE) *JIS numericDate;
            if (not %error) ;
              return *on;
            endif;
       endsl;
       return *off;

  /END-FREE
p                 E
```

Anexo F: Código de Fuente del Programa R71M0018

```
 *------------------------------------------------------------------*
 *         SOFTWARE FINANCIERO AUTOMATIZADO (SOFIA)                 *
 *         ***************************************                  *
 * MODULO       : CARTERA                                           *
 * APLICACION   : VERIFICACION DE CHECK LIST DE OPERACIONES         *
 *------------------------------------------------------------------*
 * PROG.ORIGEN  : R7102310                                          *
 *------------------------------------------------------------------*
 * AUTOR        : NELSON ABEL BARRANZUELA IMAN (NEBAR)              *
 * FECHA CREAC. : 20/10/2015                                        *
 *------------------------------------------------------------------*
 * MODIFICADO POR :                          FECHA MODIF.:          *
 * ================                          =============          *
 *------------------------------------------------------------------*
 *   COMENTARIO:                                                    *
 *   ===========                                                    *
 *   04/04/2016 - validar para que los Hipotecarios no se pida llenar*
 *                el Check List                                     *
 *------------------------------------------------------------------*
h EXTBININT(*YES)
h AlwNull(*usrctl)
HDFTACTGRP(*NO) ACTGRP(*CALLER)
hdatedit(*DMY)
hdatfmt(*EUR) timfmt(*EUR)
hBNDDIR('SRVPBND')

fP71M0018  cf   e             workstn
f                                       sfile(data01:ncon1)
f                                       indds(Dspind)
f
  /DEFINE UTILITYSQLRPG_H_DEFINED
  /COPY QRPGCPYLE,R01U0001

  /DEFINE PROPUESTA_H_DEFINED
  /COPY QRPGCPYLE,R07U0002
  /COPY SIAFF/QRPGCPYLE,R07I0039

d ncon1           s              2  0
d ncon2           s              2  0
d
 //------------------------------------------------------------------*
 //  Prototipos para llamadas                                        *
 //------------------------------------------------------------------*
 //PROGRAMA PRINCIPAL
d
d Main            pr                    Extpgm('R71M0018')
d  pemp                          40a   Const
d  pfec                           8a   Const
d  pofic                         30a   Const
d  pcodage                        3a   Const
d  psec                           3a   Const
d  pRepCre                       10  0 Const
d  pNumDes                        5  0 Const
d  pServi1                        3  0 Const
d  pmoned1                        3  0 Const
```

```
d  pMontoD                      14  2 Const
d  pCodCli                       9  0 Const
d  pNomCli                      40a   Const
d  pSalir                        1  0
d  pEnvCL                        1  0
d  nTipoExpCL                    1  0 Const
d  pFlgPresen                    2P 0 Const
d
d Main            pi
d  pemp                         40a   Const
d  pfec                          8a   Const
d  pofic                        30a   Const
d  pcodage                       3a   Const
d  psec                          3a   Const
d  pRepCre                      10  0 Const
d  pNumDes                       5  0 Const
d  pServi1                       3  0 Const
d  pmoned1                       3  0 Const
d  pMontoD                      14  2 Const
d  pCodCli                       9  0 Const
d  pNomCli                      40a   Const
d  pSalir                        1  0
d  pEnvCL                        1  0
d  nTipoExpCL                    1  0 Const
d  pFlgPresen                    2P 0 Const
d
 //-------------------------------------------------------------------*
 // Constantes                                                        *
 //-------------------------------------------------------------------*
d  true           c                   *On
d  false          c                   *Off
d  LOAN_SCHEMA    C                   Const(966)
d  PARAM_TABLE    C                   Const('F6209')
d  DOLAR_SYMBOL   C                   Const('USD')
d  SOLES_SYMBOL   C                   Const('S/.')
d  DOLAR          C                   Const(1)
d  SOLES          C                   Const(2)
d  LABEL_SOLES    C                   Const('SOLES')
d  LABEL_DOLAR    C                   Const('DOLARES')
d  DIGIDE_PARAM   C                   Const(373)
d  DIGIDE_TABLE   C                   Const('SGCRED')
 //-------------------------------------------------------------------*
 //  Indicadores de Pantalla                                          *
 //-------------------------------------------------------------------*
d  DspInd         ds
d  iSalir                        N   Overlay(DspInd:3)
d  iAyuda                        N   Overlay(DspInd:4)
d  iGrabar                       N   Overlay(DspInd:5)
d  iPendiente                    N   Overlay(DspInd:6)
d  iYesNo                        N   Overlay(DspInd:40)
d  iDspSfl                       N   Overlay(DspInd:71)
d  iFLDSPCTL                     N   Overlay(DspInd:72)
d  iFLINZar                      N   Overlay(DspInd:73)
d  iSFLCLR                       N   Overlay(DspInd:74)
d  iSFLEND                       N   Overlay(DspInd:75)
d  iFlgSN                        N   Overlay(DspInd:81)
d  iFlgViewCLP                   N   Overlay(DspInd:83)
 // ------------------------------------------------------------------*
 //        Estructura de datos de documentos/aspectos de CL           *
```

```
      // ------------------------------------------------------------------*
     d  dsAspectos     ds
     d  nCodDocum                     5  0
     d  sDescDocum                  500a
     d  nRptaSys                      1  0
     d  nOrdenDocum                   3  0
      ***************************************************
      *  Program's Data Structures                      *
      ***************************************************
     d  arrayTipoChkList...
     d                 ds                  likeds(ChekList_Data_t) dim(99)
     d dataTipChkList  ds                  likeds(ChekList_Data_t) Inz
     d dsExpChkListPro...
     d                 ds                  likeds(rowExpedChkListPropuesta)
Inz
     d expChkListDto   ds                  likeds(expedChkListPropuestaDto)
Inz
     d dsTipoExpChkList...
     d                 ds                  likeds(rowTipoExpChkList) Inz
     d dsMessaDesembolso...
     d                 ds                  likeds(rowMessageDesembolso) Inz
     d dsClienteDto    ds                  likeds(cliente_t) Inz
     d
      ***************************************************
      *  Stand Alone Variables                          *
      ***************************************************
     d  tablaDigide    s              10    Inz('F6209')
     d  digide         s               3  0 Inz(373)
     d  codTabla       s              10    Inz('SGCRED')
     d
     d
     d arrAnswerAllowNo...
     d                 s               5P 0 dim(10)
     d ANSWER_ALLOW_NO...
     d                 C                   Const(1)
     d answerNoAux     s               1  0 Inz(0)
     d prevCategory    s               2  0 Inz(-1)
     d index           s               3  0 Inz(1)
     d tabCodigo       s              19  0 INZ(12)
     d  schema         s              10    Inz('')
     d updateCorrelat  s                N   Inz(*on)
     d isQuestionAcceptNo...
     d                 s                N   Inz(*on)
     d isQuestionAcceptSI...
     d                 s                N   Inz(*on)
     d isSaveCHLPREV   s                N   Inz(*off)
     d
      //-----------------------------------------------------------------*
      //  variables  numéricas                                           *
      //-----------------------------------------------------------------*
     d  nTipoCL        s               1  0
     d  nTipoCat       s               2  0
     d  nContExpProp   s               5  0
     d  nContMsgProp   s               5  0
     d  nCodExp        s              19  0
     d  nCorr          s              19  0
     d  nContItem      s              19  0
     d  nRptaUser      s               1  0
     d  nFlgCLOk       s               1  0
```

```
     d  nFlgSit        s              1  0
     d  nFlgRes        s              1  0
      //-------------------------------------------------------------------*
      //   Variables Alfanuméricas                                         *
      //-------------------------------------------------------------------*
     d sMensaje        s            120a
     d sAntesComen     s            250a
     d sIndice         s              1a
     d  sCodDNI        s              1a
     d sFecha          s             26a
      //-------------------------------------------------------------------*
      //   Variables Fecha y Hora                                          *
      //-------------------------------------------------------------------*
     d zFecRem         s               z
     d
      * Fecha actual
     D                 DS                  Inz
     D nFecCal                 1      8  0
     D    nDiaCal              1      2  0
     D    nMesCal              3      4  0
     D    nAnioCal             5      8  0

      /FREE
       Exec SQL
        set option commit=*none,datfmt=*iso;
       Exec SQL
       set path=*libl;
       //-------------------------------------------------------------------*
       //  RUTINA PRINCIPAL                                                 *
       //-------------------------------------------------------------------*
         tablaDigide=PARAM_TABLE;
         digide= LOAN_SCHEMA;
         codTabla=%EDITC(pServi1:'X');
         schema = getDesLarFromDigide(tablaDigide:digide:codTabla);
         if schema='';
            nFecCal = %dec(pfec:8:0);
            sFecha=%EDITC(nAnioCal:'X')+'-'+%EDITC(nMesCal:'X')+'-'+
                   %EDITC(nDiaCal:'X')+'-'+
                   %EDITC(%dec(%SUBDT(%TIME():*H):2:0):'X')+'.'+
                   %EDITC(%dec(%SUBDT(%TIME():*MN):2:0):'X')+'.'+
                   %EDITC(%dec(%SUBDT(%TIME():*S):2:0):'X')+'.000000';
            zFecRem=%Timestamp(sFecha);
            if nTipoExpCL=CHK_DURANTE_DESEMBOLOSO and
pFlgPresen=NO_PRESENCIAL;
               dsMessaDesembolso.REPCRE = pRepCre;
               dsMessaDesembolso.NUMDES = pNumDes;
               dsMessaDesembolso.USULEV = systemUser;
               dsMessaDesembolso.DIALEV = nDiaCal;
               dsMessaDesembolso.MESLEV = nMesCal;
               dsMessaDesembolso.AÑOLEV = nAnioCal;
               dsMessaDesembolso.OBSERV = %trim(WANTCOM);

               updateMessageDesembolso(dsMessaDesembolso);

               exsr sbrfin;
            endif;
            exsr sbrProcesa;
            exsr sbrValidaExpCL;
         else;
```

```
            pEnvCL=2;
          endif;
          exsr sbrfin;
        //-------------------------------------------------------------------*
        // Subrutina Procesa Operaciones                                     *
        //-------------------------------------------------------------------*
        begsr sbrProcesa;

           exsr sbrCargarParam;
           iSalir= false;
           exsr sbrObtenerCodExp;
           dow not iSalir;
             if nTipoExpCL=CHK_PREVIO_DESEMBOLSO;
                iFlgViewCLP=true;
             else;
                iFlgViewCLP=false;
             endif;
             WRITE  F00;
             exfmt  CONTRO01;
             select;
               when iSalir;
                 leave;
               when iPendiente;
                 sMensaje='¿Desea guardar el expediente Check List (CL) como'+
                          ' pendiente?';
                 if ProcMsgBox('Pregunta':sMensaje:'05')='06';
                    exsr sbrGrabarExpParcial;
                    iSalir=true;
                    pSalir=1;
                 endif;
               when iAyuda;
                   exsr sbrImprimeDetalle;
               other;
                 exsr sbrValidaExpCL;
                 if nFlgCLOk = 1;
                    if isQuestionAcceptSI=*on and isQuestionAcceptNo=*on;
                       exsr sbrMostrarLlenadoCL;
                    else;
                       sMensaje='Para continuar debe Responder correctamente '
+
                                'las Preguntas';
                       ProcMsgBox('Error':sMensaje:'10');
                    endif;
                 else;
                    sMensaje='Para continuar debe verificar el llenado del ' +
                             'Check List: SI=1 ó NO=0';
                    ProcMsgBox('Error':sMensaje:'10');
                 endif;
             endsl;
           enddo;

        endsr;

        //-------------------------------------------------------------------*
        // Carga parámetros en pantalla                                      *
        //-------------------------------------------------------------------*
        begsr sbrCargarParam;
          iFLINZar=true;
          write CONTRO01;
```

```
  iFLINZar=false;
  iDspSfl = false;
  iFLDSPCTL = false;
  iSFLCLR = true;
  if nTipoExpCL=CHK_PREVIO_DESEMBOLSO;
     iFlgViewCLP=true;
  else;
     iFlgViewCLP=false;
  endif;
  //nFecCal = *Date;
  nFlgRes=0;
  WEMP=pemp;
  WOFIC=pofic;
  WSEC=psec;
  WFEC= %dec(pfec:8:0);
  WSERVI1= pServi1;
  WCODCLI=pCodCli;
  WNOMCLI=pNomCli;
  WPROPNUM=pRepCre;
  MONTOD=pMontoD;
  If pmoned1 = SOLES ;
     WMONEDA = SOLES_SYMBOL;
     WMONED1= LABEL_SOLES;
  ElseIf pmoned1=DOLAR;
        WMONEDA = DOLAR_SYMBOL;
        WMONED1= LABEL_DOLAR;
  // Others currencies
  EndIf;
  // Getting Customer's Data
  dsClienteDto = getClienteDto(pCodCli);
  ExSr sbrMostrarExpAntes;
EndSr;

// ------------------------------------------------------------------*
//     valida Llenado de Check List
// ------------------------------------------------------------------*
begsr sbrMostrarLlenadoCL;
    WFEC2=WFEC;
     if nFlgSit=2;
        WANTEXP='0';
     else;
        if nFlgSit=3;
          WANTEXP='1';
        endif;
     endif;
    WANTCOM=sAntesComen;
    iFlgViewCLP=false;
    dow not iSalir;
     write F01;
     exfmt F01;
     select;
      when iSalir;
        iSalir= false;
        leave;
      when iGrabar;
         exsr sbrValidaPreguntaAntes;
         if nFlgCLOk=1;
            if nTipoCL=CHK_PREVIO_DESEMBOLSO;
               sMensaje ='¿Desea grabar lo verificado?';
```

```
 else;
    if nTipoCL=CHK_DURANTE_DESEMBOLOSO;
       sMensaje ='¿Desea continuar con el desembolso?';
    endif;
 endif;

If ProcMsgBox('Pregunta':sMensaje:'05')='06';
    pEnvCL=1;
    ExSr sbrGrabarExp;
 If nTipoCL=CHK_PREVIO_DESEMBOLSO;
  nContMsgProp = getNumberMessagePropuesta(pRepCre:pNumDes)
                    + 1;
  If WANTEXP='1';
      // Ckeck list Previo se envió OK y graba un mensaje'
        // de restricción durante el desembolso
       sMensaje='CHECK LIST DURANTE DESEMB. PROPUESTA '+
                %char(pRepCre)+' CLIENTE '+%char(pCodCli);
       nFlgRes=2;

       dsMessaDesembolso.REPCRE = pRepCre;
       dsMessaDesembolso.NUMDES = pNumDes;
       dsMessaDesembolso.CORREL = nContMsgProp;
       dsMessaDesembolso.MENSAJ = sMensaje;
       dsMessaDesembolso.DIAREG = nDiaCal;
       dsMessaDesembolso.MESREG = nMesCal;
       dsMessaDesembolso.AÑOREG = nAnioCal;
       dsMessaDesembolso.FLGRES = nFlgRes;
       dsMessaDesembolso.USUREG = systemUser;

       saveMessageDesembolso(dsMessaDesembolso);
       isSaveCHLPREV = *on;
       iSalir= true;
     Else;
        //envia mensaje de alerta
        sMensaje='CHECK LIST PREVIO PROPUESTA:'+
             %char(pRepCre)+'. CLIENTE:'+%char(pCodCli);
       nFlgRes=2;
       dsMessaDesembolso.REPCRE = pRepCre;
       dsMessaDesembolso.NUMDES = pNumDes;
       dsMessaDesembolso.CORREL = nContMsgProp;
       dsMessaDesembolso.MENSAJ = sMensaje;
       dsMessaDesembolso.MONTOR = pMontoD;
       dsMessaDesembolso.MONEDA = pmoned1;
       dsMessaDesembolso.DIAREG = nDiaCal;
       dsMessaDesembolso.MESREG = nMesCal;
       dsMessaDesembolso.AÑOREG = nAnioCal;
       dsMessaDesembolso.FLGRES = nFlgRes;
       dsMessaDesembolso.USUREG = systemUser;

       saveMessageDesembolso(dsMessaDesembolso);

     iSalir= true;
     EndIf;
   Else;
     if nTipoCL=CHK_DURANTE_DESEMBOLOSO;
       if WANTEXP='1';
        dsMessaDesembolso.REPCRE = pRepCre;
        dsMessaDesembolso.NUMDES = pNumDes;
        dsMessaDesembolso.USULEV = systemUser;
```

```
                        dsMessaDesembolso.DIALEV = nDiaCal;
                        dsMessaDesembolso.MESLEV = nMesCal;
                        dsMessaDesembolso.AÑOLEV = nAnioCal;
                        dsMessaDesembolso.OBSERV = %trim(WANTCOM);

                         updateMessageDesembolso(dsMessaDesembolso);
                       endif;
                       iSalir= true;
                    endif;
                 endif;
               endif;
            else;
               if nFlgCLOk=0;
                  sMensaje='Debe escribir un comentario';
               endif;
               if nFlgCLOk=-1;
                  sMensaje='Usted debe responder la pregunta '+
                           '(SI=1 ó NO=0)';
               endif;
               if nFlgCLOk = 3;
                  sMensaje = 'Respuestas del Check List no concuerdan '
+
                             'Revisar';
                  ProcMsgBox('Error':sMensaje:'10');
               endif;
            endif;

         other;
          exsr sbrValidaPreguntaAntes;
          if nFlgCLOk<>1;
             if nFlgCLOk=0;
                sMensaje='Debe escribir un comentario';
            endif;
            if nFlgCLOk=-1;
                 sMensaje='Usted debe responder la pregunta '+
                          '(SI=1 ó NO=0)';
            endif;
            if nFlgCLOk = 3;
              sMensaje = 'Respuestas del Check List no concuerdan ' +
                             'Revisar';
               endif;
              ProcMsgBox('Error':sMensaje:'10');
           endif;
      endsl;
   enddo;
ENDSR;

// ------------------------------------------------------------------*
//      valida siguiente pantalla de pregunta (Antes Desembolso)   *
// ------------------------------------------------------------------*
begsr sbrValidaPreguntaAntes;
    nFlgCLOk=1;
    if %trim(WANTCOM)='' ;
       nFlgCLOk=0;
    endif;
    if WANTEXP<>'1' and WANTEXP<>'0';
       nFlgCLOk=-1;
    ENDIF;
    if nTipoCL = CHK_PREVIO_DESEMBOLSO;
```

```
      if WANTEXP = '1' ;
        if sCodDNI='0';
            nFlgCLOk = 3;
        ENDIF;
      ENDIF;
      if WANTEXP = '0' ;
        if sCodDNI='1';
            nFlgCLOk = 3;
        ENDIF;
      ENDIF;
    endif;
    if nTipoCL = CHK_DURANTE_DESEMBOLOSO;
        if WANTEXP = '1' and isQuestionAcceptSI = *on
             and isQuestionAcceptNo = *off;
           nFlgCLOk=3;
        endif;
        if WANTEXP = '0' and isQuestionAcceptSI = *on and
          isQuestionAcceptNo = *on;
            nFlgCLOk=3;
        ENDIF;
    endif;

ENDSR;

// ---------------------------------------------------------------*
//              Valida Expediente Check List                      *
// ---------------------------------------------------------------*
begsr sbrValidaExpCL;
  isQuestionAcceptSI = *on;
  isQuestionAcceptNO = *on;
  ncon2 = 1;
  nFlgCLOk=1;
    chain ncon2 data01;
    dow %found();
      if WCODDOC <> 0 and WCODDOC<>84;
        if WEXPEDIEN<>'0' and WEXPEDIEN<>'1' ;
            nFlgCLOk=0;
        endif;
        //Si acepta SI y es respondida con NO
        if %lookup(WCODDOC:arrAnswerAllowNo) = 0 and WEXPEDIEN = '0';
                isQuestionAcceptSI = *off;
          endif;
          //SI acepta NO como respuesta y se responde con SI
         if %lookup(WCODDOC:arrAnswerAllowNo) > 0;
            if WEXPEDIEN <> '0';
               isQuestionAcceptNo = *off;
            endif;
         endif;
       else;
          if WCODDOC=84;
            sCodDNI=WEXPEDIEN;
          endif;
       endif;
     ncon2+=1;
     chain ncon2 data01;
    enddo;
endsr;

// ---------------------------------------------------------------*
```

```
//               Graba Expediente Total Check List                *
// ---------------------------------------------------------------*
begsr sbrGrabarExp;
    if WANTEXP = '0';
       nFlgSit = 2;
    else;
       if WANTEXP = '1';
          nFlgSit = 3;
       endif;
    endif;
    clear dsExpChkListPro; //Limpiando estructura
    // Asiganando Valores
    dsExpChkListPro.FLGSIT = nFlgSit;
    dsExpChkListPro.USRMOD = systemUser;
    dsExpChkListPro.COMREV = %trim(WANTCOM);
    dsExpChkListPro.FECMOD = zFecRem;
    dsExpChkListPro.USRREV = systemUser;
    dsExpChkListPro.FECREV = zFecRem;
    dsExpChkListPro.PROPNRO = pRepCre;
    dsExpChkListPro.TIPCHK = nTipoCL;

    updateExpedCheckListPropuesta(dsExpChkListPro);

    exsr sbrGrabarExpParcial;
endsr;

// ---------------------------------------------------------------*
//             Graba Parcialmente Expediente Check List           *
// ---------------------------------------------------------------*
BegSr sbrGrabarExpParcial;

    clear dsTipoExpChkList;
    //Asignando Valores
    dsTipoExpChkList.COEXPRO = nCodExp;
    dsTipoExpChkList.PROPNRO = pRepCre;
    dsTipoExpChkList.TIPEXP = nTipoCL;
    deleteTipoExpedCheckList(dsTipoExpChkList);

    ncon2 = 1;
    chain ncon2 data01;
    Dow %found();
       nRptaUser=0;
       If WEXPEDIEN='0';
          nRptaUser=4;
       Else;
          If WEXPEDIEN='1' ;
             nRptaUser=5;
          EndIf;
       EndIf;
       If WCODDOC <> 0;
          Clear dsTipoExpChkList;
          dsTipoExpChkList.COEXPRO = nCodExp;
          dsTipoExpChkList.PROPNRO = pRepCre;
          dsTipoExpChkList.TIPEXP = nTipoCL;
          dsTipoExpChkList.CODDOC = WCODDOC;
          dsTipoExpChkList.RPTCHK = nRptaUser;
          dsTipoExpChkList.USRREG = systemUser;
          dsTipoExpChkList.FECREG = zFecRem;
```

```
            saveTipoExpedCheckList(dsTipoExpChkList);

          EndIf;
          ncon2+=1;
          chain ncon2 data01;
        EndDo;
ENDSR;

//-------------------------------------------------------------------*
// Muestra el Expediente Check List Operaciones (Antes desembolso) *
//-------------------------------------------------------------------*
begsr sbrMostrarExpAntes;
  WRITE CONTRO01;
  iSFLCLR=false;
  exsr sbrLlenaCL;
  iSFLEND = true;
  iDspSfl = true;
  iFLDSPCTL = true;
ENDSR;

//-------------------------------------------------------------------*
//                  Llena Check List de Operaciones                  *
//-------------------------------------------------------------------*
BegSr sbrLlenaCL;

   nTipoCL = nTipoExpCL;
   ncon1 = 0;
   arrayTipoChkList = getDataChecklist(pRepCre:nTipoExpCL
                                       :dsClienteDto.clasePersona);

   Dow arrayTipoChkList(index) <> *Blanks;
       dataTipChkList = arrayTipoChkList(index);
       If  dataTipChkList.answerAllowNo = ANSWER_ALLOW_NO;
          arrAnswerAllowNo(index) = dataTipChkList.codigoDoc;
       EndIf;
       If (prevCategory <> dataTipChkList.kindCategory);
          ncon1 = ncon1 + 1;
          iFlgSN = true;
          WASPECTO = dataTipChkList.deslap;
          WCODDOC = 0;
          Write DATA01;
          ncon1 = ncon1 + 1;
          iFlgSN = false;
          WASPECTO = %char(dataTipChkList.indice) + '.'
                     + %trim(dataTipChkList.descripcion);
          WEXPEDIEN = %char(dataTipChkList.systemanswer);
          WCODDOC = dataTipChkList.codigoDoc;
          Write DATA01;
       Else;
          ncon1 = ncon1 + 1;
          iFlgSN = false;
          WASPECTO = %char(dataTipChkList.indice) + '.'
                     + %trim(dataTipChkList.descripcion);
          WEXPEDIEN = %char(dataTipChkList.systemanswer);
          WCODDOC = dataTipChkList.codigoDoc;
          Write DATA01;
       EndIf;
       prevCategory = dataTipChkList.kindCategory;
       index += 1;
```

```
   EndDo;
EndSr;
//-----------------------------------------------------------------*
//Obtener código Expediente check List (CORRELATIVOS: TABCODIGO=12)*
//-----------------------------------------------------------------*
begsr sbrObtenerCodExp;

    If (Not existExpedCheckListProp(pRepCre:nTipoCL:1));
      tablaDigide=PARAM_TABLE;
      digide = DIGIDE_PARAM;
      codTabla=DIGIDE_TABLE;
      schema = getDesLarFromDigide(tablaDigide:digide:codTabla);
      Monitor;
         addLibLE(%trim(schema));
      On-Error;
      EndMon;

    nCodExp = getNextValueFromCorrelativos(tabCodigo:updateCorrelat);
      //Fill structure's data
    dsExpChkListPro.COEXPRO = nCodExp;
    dsExpChkListPro.PROPNRO = pRepCre;
    dsExpChkListPro.NUMDES = pNumDes;
    dsExpChkListPro.TIPCHK = nTipoCL;
    dsExpChkListPro.FLGSIT = 0;
    dsExpChkListPro.FLGEST = 1;
    dsExpChkListPro.USRREG = systemUser;
    dsExpChkListPro.FECREG = zFecRem;

     saveExpedCheckListPropuesta(dsExpChkListPro);

    Else;
      expChkListDto = getExpedCheckListDto(pRepCre:nTipoCL);
      nCodExp = expChkListDto.codigoProuesta;
      nFlgSit = expChkListDto.flagSituacion;
      sAntesComen = expChkListDto.comentario;
    EndIf;

EndSr;

//-----------------------------------------------------------------*
// Finaliza Programa                                              *
//-----------------------------------------------------------------*
begsr sbrFin;
  *inLR=true;
  return;
endsr;

//-----------------------------------------------------------------*
// Finaliza Programa                                              *
//-----------------------------------------------------------------*
begsr sbrImprimeDetalle;
  chain $NEW$CRRN$ DATA01;
  if %Found();
     monitor;
        index=%dec(%subst(wAspecto:1:1):1:0);
        DESCRI1=%subst(arrayTipoChkList(index):9:50);
        DESCRI2=%subst(arrayTipoChkList(index):59:50);
        DESCRI3=%subst(arrayTipoChkList(index):109:50);
        DESCRI4=%subst(arrayTipoChkList(index):159:50);
```

```
         exfmt w01;
       on-error;
       endmon;
    endif;
    iSalir=false;
 endsr;
/END-FREE
/COPY SIAFF/QRPGCPYLE,R07P0039
```